Algoritmos de Broadcast e Multicast de Ordem Total: Taxonomia e Pesquisa

XAVIER DEFAGO

Instituto Avançado de Ciência e Tecnologia do Japão e PRESTO, Agência Japonesa de Ciência e Tecnologia

ANDRE' SCHIPER

Escola Politécnica Federal de Lausanne, Suíça

E

PETER URBUM

Instituto Avançado de Ciência e Tecnologia do Japão

O broadcast de ordem total e o multicast (também chamado de broadcast/multicast atômico) apresentam um problema importante em sistemas distribuídos, especialmente no que diz respeito à tolerância a falhas. Resumindo, a primitiva garante que as mensagens enviadas para um conjunto de processos sejam, por sua vez, entregues por todos esses processos na mesma ordem total.

Categorias e Descritores de Assunto: C.2.4 **[Redes de Comunicação de Computadores]:** Sistemas Distribuídos; C.2.2 **[Redes de Comunicação de Computadores]:** Protocolos de Rede —*Aplicações; Arquitetura de protocolo;* D.4.4 **[Sistemas Operacionais]:** Gerenciamento de Comunicações— *Envio de mensagens; Comunicação de rede;* D.4.5 **[Sistemas operacionais]:** *Confiabilidade—Tolerância a falhas;* H.2.4 **[Gerenciamento de banco de dados]:** *Sistemas—Bancos de dados distribuídos;* H.3.4 **[Armazenamento e recuperação de informações]:** Sistemas e *software—Sistemas distribuídos*

Termos Gerais: Algoritmos, Confiabilidade, Design

Palavras-chave e frases adicionais: Sistemas distribuídos, algoritmos distribuídos, comunicação em grupo, tolerância a falhas, problemas de concordância, passagem de mensagens, ordenação total, ordenação global, multicast atômico, transmissão atômica, classificação, taxonomia, pesquisa

---

Parte desta pesquisa foi conduzida para o programa “Promovendo Talentos em Campos de Pesquisa Emergentes” em Fundos de Coordenação Especial para Promoção de Ciência e Tecnologia pelo Ministério de Educação, Cultura, Esportes, Ciência e Tecnologia do Japão. Este trabalho foi iniciado durante o doutorado de Xavier Defago. pesquisa no Instituto Federal Suíço de Tecnologia em Lausanne [Defago 2000]. P eter Urb ´an foi apoiado pela Sociedade Japonesa para a Promoção da Ciência, um subsídio para bolsistas JSPS do Ministério da Educação, Cultura, Esportes, Ciência e Tecnologia do Japão, das Fundações Nacionais de Ciência da Suíça e do CSEM Centro Suíço de Eletrônica e Microtecnologia, Inc., Neuchatel. ˆ Endereços dos autores:

X. Defago e P. Urb an, School of Information Science, JAIST, 1-1 Asahidai, Tatsunokuchi, Nomigun, Ishikawa 923-1292, Japão; e-mail: {defago,urban}@jaist.ac.jp; A. Schiper, IC LSR, Escola de Informação e Comunicação, EPFL, CH-1015 Lausanne, Suíça; e-mail: André. Schiper@epfl.ch.

O problema inspirou uma abundância de literatura, com uma infinidade de algoritmos propostos. Este artigo propõe uma classificação de algoritmos de broadcast e multicast de ordem total com base em seus mecanismos de ordenação e aborda uma série de outras questões importantes. O artigo examina cerca de sessenta algoritmos, fornecendo, de longe, o estudo mais extenso do problema até agora. O artigo discute algoritmos para modelos de sistemas síncronos e assíncronos e estuda as respectivas propriedades e comportamento dos diferentes algoritmos.

1. INTRODUÇÃO

Sistemas e aplicativos distribuídos são notoriamente difíceis de construir. Isso se deve principalmente à inevitável simultaneidade em tais sistemas, combinada com a dificuldade de fornecer um controle global. Essa dificuldade é bastante reduzida ao confiar em primitivos de comunicação em grupo que fornecem garantias mais altas do que a comunicação ponto a ponto padrão.

Uma dessas primitivas é chamada de broadcast ordem1 total.2 Informalmente, a primitiva garante que as mensagens enviadas para um conjunto de processos sejam entregues por todos esses processos na mesma ordem. A transmissão de ordem total é uma primitiva importante que desempenha um papel central, por exemplo, ao implementar a abordagem da máquina de estado (também chamada de replicação ativa) [Lamport 1978a; Schneider 1990; Poledna 1994]. Também possui outras aplicações, como sincronização de relógio [Rodrigues et al. 1993], o computador suportou a escrita cooperativa, distribuiu a memória compartilhada e distribuiu a exclusão mútua [Lamport 1978b]. Mais recentemente, também foi demonstrado que um uso adequado de transmissão de ordem total pode melhorar significativamente o desempenho de bancos de dados replicados [Agrawal et al. 1997; Pedone et ai. 1998; Kemme et al. 2003].

Literatura em transmissão de ordem total. Existe uma quantidade considerável de literatura sobre transmissão de ordem total, e muitos algoritmos, seguindo várias abordagens, foram propostos para resolver este problema. É, no entanto, difícil compará-los, pois muitas vezes diferem em relação às suas propriedades reais, suposições, objetivos ou outros aspectos importantes. Portanto, é difícil saber qual solução é mais adequada para um determinado contexto de aplicativo.

Quando confrontado com novos requisitos, a ausência de um roteiro para o problema de transmissão de ordem total pode levar engenheiros e pesquisadores a desenvolver novos algoritmos em vez de adaptar soluções existentes (reinventando assim a roda), ou usar uma solução pouco adequada para o problema. necessidades de aplicação. Um passo importante para melhorar a situação atual é fornecer uma classificação dos algoritmos existentes.

Trabalho relatado. Tentativas anteriores foram feitas para classificar e comparar algoritmos de transmissão de ordem total [Anceaume 1993b; Anceaume e Minet 1992; Cristiano e cols. 1994; Friedman e van Renesse 1997; Maier 1992]. No entanto, nenhum é baseado em uma pesquisa abrangente dos algoritmos existentes e, portanto, todos carecem de generalidade.

A comparação mais completa até agora foi feita por Anceaume e Minet [1992] (uma versão estendida foi posteriormente publicada em francês por Anceaume [1993b]), que adotam uma abordagem interessante baseada nas propriedades dos *algoritmos* . Seu artigo levanta algumas questões fundamentais que inspiraram uma parte do nosso trabalho. É, no entanto, um pouco desatualizado agora. Além disso, os autores estudam apenas sete algoritmos diferentes, que não são verdadeiramente representativos; por exemplo, nenhum é baseado em uma abordagem de histórico de comunicação (uma das cinco classes de algoritmos; detalhes na Seção 4.4).

1A transmissão de ordem total também é conhecida como difusão atômica. Ambas as terminologias estão atualmente em uso. Há uma ligeira controvérsia com relação ao uso de um sobre o outro. Optamos pelo primeiro, ou seja, “total ou der broadcast”, porque o último é um pouco enganador. De fato, a atomicidade sugere uma propriedade relacionada à concordância e não à ordem total (definida na Seção 2), e a ambigüidade já foi fonte de mal-entendidos. Em contraste, “total or der broadcast” refere-se inequivocamente à propriedade da ordem total.

2O *multicast* de ordem total às vezes é usado em vez da *transmissão de ordem total.* A distinção entre os dois primitivos é explicada mais adiante no artigo (Seção 3). Quando a distinção não é importante, usamos o termo *broadcast de ordem total.*

Cristiano e cols. [1994] adotam uma abordagem diferente, focando na implementação dos algoritmos, ao invés de suas propriedades. Eles estudam quatro algoritmos diferentes e os comparam usando simulação de eventos discretos. Eles encontram resultados interessantes sobre o respectivo desempenho de diferentes estratégias de implementação. No entanto, eles falham em discutir as respectivas propriedades dos diferentes algoritmos.
Além disso, como comparam apenas quatro algoritmos, este trabalho é menos geral que o de Anceaume [1993b].

Friedman e van Renesse [1997] estudam o impacto do empacotamento de mensagens no desempenho de algoritmos. Para tanto, estudam seis algoritmos, incluindo os estudados por Cristian et al. [1994].
Eles medem o desempenho real dos algoritmos e confirmam as observações feitas por Cristian et al. [1994]. Eles mostram que o empacotamento de várias mensagens de protocolo em uma única mensagem física fornece uma maneira eficaz de melhorar o desempenho dos algoritmos. A comparação também carece de generalidade, mas isso é bastante compreensível, pois essa não é a principal preocupação do artigo.

Mayer [1992] define uma estrutura na qual os algoritmos de transmissão de ordem total podem ser comparados do ponto de vista do desempenho. A definição de tal estrutura é um passo importante em direção a uma comparação extensa e significativa de algoritmos.
No entanto, o artigo não compara os numerosos algoritmos existentes.

Contribuições. Neste artigo, propomos uma classificação de algoritmos de broadcast de ordem total com base no mecanismo usado para ordenar as mensagens. A razão para esta escolha é que o mecanismo de ordenação é a característica com maior influência no padrão de comunicação do algoritmo: dois algoritmos da mesma classe provavelmente exibirão comportamentos semelhantes. Definimos cinco classes de mecanismos de ordenação: *histórico de comunicação, baseado em privilégios, seqüenciador móvel, seqüenciador fixo* e *acordo de destinos.*

Neste artigo, também fornecemos uma vasta pesquisa de cerca de sessenta algoritmos de transmissão total ou der publicados. Sempre que possível, mencionamos as propriedades e os hipóteses de cada algoritmo. Isso, no entanto, nem sempre é possível porque as informações disponíveis nos artigos muitas vezes não são suficientes para caracterizar com precisão o comportamento do algoritmo (por exemplo, diante de uma falha).

Estrutura. O artigo está logicamente organizado em quatro partes principais: especificação, mecanismos de ordenação e taxonomia, tolerância a falhas e pesquisa. Mais precisamente, o artigo está estruturado da seguinte forma. A Seção 2 apresenta a especificação do problema de broadcast de ordem total (também conhecido como broadcast atômico). A Seção 3 estende a especificação considerando as características dos grupos de destino (por exemplo, grupos únicos versus grupos múltiplos). Na Seção 4, definimos cinco classes de algoritmos de broadcast de ordem total, de acordo com a forma como as mensagens são ordenadas: *histórico de comunicação, baseado em privilégios, seqüenciador móvel, seqüenciador fixo* e *acordo de destinos.* A Seção 5 discute questões de modelo de sistema em relação a
A Seção 6 apresenta os principais mecanismos nos quais os algoritmos de transmissão de ordem total dependem para garantir a tolerância a falhas. A Seção 7 fornece uma ampla pesquisa dos algoritmos de transmissão ampla de ordem total encontrados na literatura. Os algoritmos são agrupados em suas respectivas classes, e discutimos suas principais características. A Seção 8 discute algumas outras questões de interesse relacionadas à transmissão de ordem total. Por fim, a Seção 9 conclui o artigo.

2. ESPECIFICAÇÃO DA TRANSMISSÃO TOTAL DO PEDIDO

Nesta seção, damos a especificação formal do problema de broadcast de ordem total. Como existem muitas variantes do problema, apresentamos aqui a especificação mais simples e discutimos outras variantes na Seção 3.

2.1. Notação

A Tabela I resume algumas das notações usadas ao longo do artigo. M é o conjunto contendo todas as possíveis mensagens válidas. denota o conjunto de todos os processos no sistema. Dado algum arbitrário

Tabela I. Conjunto de

M	notações de todas as mensagens válidas. conjunto de todos os processos do sistema.
sender(m)	remetente da mensagem m.
Dest(m)	conjunto de processos de destino para a mensagem m. conjunto de
remetente	todos os processos de envio no sistema. conjunto
começar	de todos os processos de destino no sistema.

mensagem m, *sender(m)* designa o processo no qual *m* se origina, e *Dest(m)* denota o conjunto de todos os processos de destino para m.

Além disso, *sender* é o conjunto de todos os processos que potencialmente podem enviar alguma mensagem válida.

$$remetente = \{p \mid p \text{ pode enviar alguma mensagem } m \in M\}. \quad (1)$$

Da mesma forma, *dest* é o conjunto de todos os destinos potenciais de mensagens válidas.

$$dest \stackrel{def}{=} \bigcup_{m \in M} Dest(m). \quad (2)$$

2.2. Falhas de processo

A especificação do amplo elenco de ordem total requer a definição da noção de um processo *correto* . O seguinte conjunto de classes de falha de processo é comumente considerado:

— *Falhas de colisão.* Quando um processo trava, ele deixa de funcionar para sempre. Isso significa que ele para de realizar qualquer atividade, incluindo enviar, transmitir ou receber qualquer mensagem.

— *Falhas de omissão.* Quando um processo falha por omissão, ele deixa de realizar algumas ações, como enviar ou receber uma mensagem.

— *Falhas de temporização.* Uma falha de temporização ocorre quando um processo viola algumas das suposições de temporização do modelo do sistema (detalhes na Seção 5.1). Obviamente, este tipo de falha não existe em modelos de sistemas assíncronos, devido à ausência de suposições de tempo em tais sistemas.

— *Falhas bizantinas.* As falhas bizantinas são o tipo mais geral de falhas. A

O componente bizantino tem permissão para qualquer comportamento arbitrário. Por exemplo, um processo defeituoso pode alterar o conteúdo das mensagens, duplicar mensagens, enviar mensagens não solicitadas ou mesmo tentar, de forma maliciosa, interromper todo o sistema.

Um processo *correto* é definido como um processo que nunca expressa nenhum dos comportamentos defeituosos mencionados acima.

2.3. Especificação Básica da Transmissão Total de Pedidos

Agora podemos fornecer a especificação mais simples da transmissão total da ordem. Formalmente, o problema é definido em termos de duas primitivas, chamadas *TO-broadcast(m)* e *TO-deliver(m),* onde *m* ÿ M é alguma mensagem. Quando um processo *p* executa *TO-broadcast(m)* (respectivamente *TO-deliver(m)),* podemos dizer que *p* TO transmite *m* (respectivamente TO-deliver m). Assumimos que toda mensagem *m* pode ser identificada de forma única e carrega a identidade de seu remetente, denotada por *sender(m).* Além disso, assumimos que, para qualquer mensagem m, e qualquer execução, *TO-broadcast(m)* é executado no máximo uma vez. Nesse contexto, a transmissão de ordem total é definida pelas seguintes propriedades [Hadzilacos e Toueg 1994; Chandra e Toueg 1996]:

(VALIDADE) Se um processo correto TO transmite uma mensagem m, então ele eventualmente entrega TO- m.

(ACORDO UNIFORME) Se um processo TO entrega uma mensagem m, então todos os processos corretos eventualmente entregam TO- m.

(UNIFORM INTEGRITY) Para qualquer mensagem m, todo processo TO-entrega *m* no máximo uma vez, e somente se *m* foi previamente transmitido TO pelo *remetente(m).*

(ORDEM TOTAL UNIFORME) Se os processos *p* e *q* ambos TO-entregam mensagens *m* e *m* então , *p* TO-entrega *m* antes de *m* se e somente,se *q* TO-entrega *m* antes de *m* .

Uma primitiva de broadcast que satisfaz todas essas propriedades, exceto Uniform Total Order (ou seja, que não oferece nenhuma garantia de ordenação) é chamada de *broadcast confiável.*

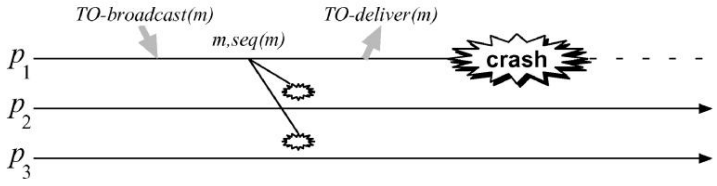


Fig. 1. Violação do Acordo Uniforme (exemplo).

Validade e Acordo Uniforme são propriedades de vivacidade. Grosso modo, isso significa que, em qualquer ponto no tempo, não importa o que tenha acontecido até aquele ponto, ainda é possível que a propriedade se mantenha [Charron-Bost et al.

2000]. Integridade uniforme e Ordem total uniforme são propriedades de segurança. Isso significa que, se, em algum ponto no tempo, a propriedade não for mantida, não importa o que aconteça depois, a propriedade não poderá ser mantida.

2.4. Propriedades não uniformes

Na definição acima de ordem total ampla, as propriedades de Acordo e Ordem Total são *uniformes.* Isso significa que essas propriedades não se aplicam apenas aos processos corretos, mas também aos defeituosos. Por exemplo, com Uniform Total Order, um processo não pode entregar nenhuma mensagem fora de ordem, mesmo que seja defeituosa. Por outro lado, a Ordem Total (não uniforme) aplica-se apenas a processos corretos e, portanto, não impõe nenhuma restrição ao comportamento de falhas.

processos.

As propriedades uniformes são fortes garantias que podem facilitar a vida dos desenvolvedores de aplicativos. Entretanto, nem todas as aplicações precisam de uniformidade, e impor a uniformidade muitas vezes tem um custo. Por esse motivo, também é importante considerar problemas mais fracos especificados usando propriedades não uniformes, embora propriedades não uniformes possam levar a inconsistências no nível da aplicação. No entanto, um aplicativo pode se proteger da não uniformidade votando (por exemplo, dado um aplicativo que coleta respostas dos destinos de uma transmissão de ordem total, o aplicativo pode votar nas respostas recebidas e considerar uma resposta efetiva somente após receber a mesma resposta da maioria). Nonuni

o Contrato de formulário e o Pedido Total são especificados da seguinte forma:

(ACORDO) Se um processo TO ***correto*** entrega uma mensagem m, então todos os processos corretos eventualmente entregam TO- m.

(ORDEM TOTAL) Se dois processos ***corretos*** *p* e *q* ambos TO-entregam mensagens *m* e então *p* TO- *m* , entrega *m* antes de *m* se e somente se *q* T,O- entrega *m* antes de *m* .

As combinações de propriedades uniformes e não uniformes definem quatro especificações diferentes para o problema de transmissão de ordem total tolerante a falhas. Essas definições constituem uma hierarquia de problemas, conforme amplamente discutido por Wilhelm e Schiper [1995]. No entanto, para simplificar, dizemos que um algoritmo de transmissão de ordem total é uniforme quando satisfaz tanto o Acordo Uniforme quanto a Ordem Total Uniforme, e dizemos que um algoritmo não é uniforme quando não impõe nenhum dos dois (ou seja, apenas suas contrapartes não uniformes). Não damos nenhum nome especial às duas definições híbridas.

A Figura 1 ilustra uma violação da propriedade Acordo Uniforme com um exemplo simples. Neste exemplo, o sequenciador p1 envia uma mensagem m, usando broadcast de ordem total. Ele primeiro atribui um número de sequência a m, depois envia *m* para todos os processos e, finalmente, entrega m. O processo p1 trava logo depois e nenhum outro processo recebe *m* (devido à perda de mensagem).

Como resultado, nenhum processo correto (por exemplo, p2) será capaz de entregar m. Acordo uniforme é violado, mas não acordo (forma não uniforme): nenhum processo *correto* entrega *m (p1* não está correto).

Nota 1. Falhas bizantinas e uniformidade. Algoritmos tolerantes a falhas bizantinas não podem garantir nenhuma das propriedades uniformes dadas na Seção 2.3.
Isso é compreensível, pois nenhum comportamento

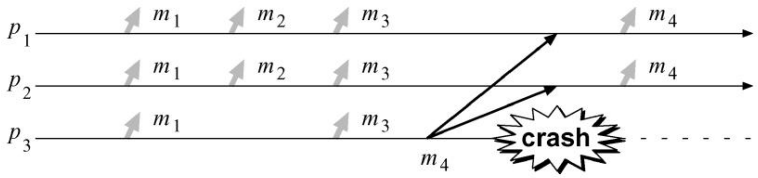


Fig. 2. Contaminação de processos corretos (p1, p2), por uma mensagem (m4), com base em um estado inconsistente *(p3* entregou m3 mas não m2).

pode ser aplicada em processos bizantinos.
Em outras palavras, nada pode impedir que um processo bizantino (1) entregue uma mensagem mais de uma vez (viola a integridade uniforme), (2) entregue uma mensagem que não é entregue por outros processos (viola o acordo) ou (3) entregue duas mensagens na ordem errada (viola a Ordem Total).

Reiter [1994] propõe uma definição mais útil de uniformidade para sistemas bizantinos. Ele distingue entre falhas e falhas bizantinas. Ele diz que um processo é *honesto* se ele se comporta de acordo com sua especificação, e *corrupto* caso contrário (isto é, bizantino), onde processos honestos também podem falhar por travamento. Nesse contexto, as propriedades uniformes são aquelas impostas por todos os processos honestos, independentemente de serem corretos ou não. Esta definição é mais sensata do que a definição mais estrita da Seção 2.3, pois nada é exigido de processos corruptos.

Nota 2. Segurança/vivacidade e uniformidade. Charron-Bost et al. [2000] mostraram que, no contexto de falhas, algumas propriedades não uniformes que comumente se acredita serem propriedades de segurança são, na verdade, propriedades de vivacidade. Eles propuseram refinamentos do conceito de segurança e vivacidade que evitam a classificação contraintuitiva.

2.5. Contaminação

O problema da contaminação vem da observação de que, mesmo com a especificação mais forte (ou seja, com Acordo Uniforme e Ordem Total Uniforme), a transmissão total de ordem não evita que um processo defeituoso *p* alcance um estado inconsistente antes de travar. Este é um problema sério porque *p* pode "legalmente"

PARA-transmitir uma mensagem com base nisso estado inconsistente e, assim, *contaminar* os processos corretos [Gopal e Toueg 1991; Anceaume e Minet 1992; Anceaume 1993b; Hadzilacos e Toueg 1994].

2.5.1. Ilustração. A Figura 2 ilustra um exemplo [Charron-Bost et al. 1999; Hadzilacos e Toueg 1994] em que um processo incorreto contamina os processos corretos. O processo p3 entrega as mensagens m1 e m3, mas não m2. Portanto, seu estado é inconsistente quando ele faz multicast m4 para os outros processos antes de travar. Os processos corretos p1 e p2 entregam m4, tornando-se assim contaminados pelo estado consistente de p3. É importante enfatizar novamente que a situação representada na Figura 2 satisfaz até mesmo a especificação mais forte apresentada até agora.

2.5.2. Especificação. É possível estender ou reformular a especificação da transmissão de ordem total de forma a impedir a contaminação. A solução consiste em impedir que qualquer processo entregue uma mensagem que possa levar a um estado in consistente.

Aguilera et ai. [2000] propõem uma reformulação do Uniform Total Order que, ao contrário da definição tradicional, não é suscetível à contaminação, pois não permite lacunas na sequência de entrega:
(GAP-FREE UNIFORM TOTAL
ORDER) Se algum processo entregar a mensagem *m* após a mensagem m, então um processo entrega *m* somente após ter entregado m.

Como alternativa, uma formulação mais antiga usa o histórico de entrega e exige que, para quaisquer dois processos dados, o histórico de um seja um prefixo do histórico do outro. Isso é expresso pela seguinte propriedade [Anceaume e Minet 1992; Cristiano e cols. 1994; Keidar e Dolev 2000]:

(ORDEM DE PREFIXO) Para quaisquer dois processos *p* e q, *hist(p)* é um prefixo de hist(q) ou *hist(q)* é um prefixo de hist(p), onde *hist(p)* e *hist(q)* são as sequências de mensagens entregues por *p* e q, respectivamente.

Nota 3. A especificação do broadcast de ordem total usando Prefix Order impede a junção dinâmica de processos (por exemplo, com uma associação de grupo). Isso pode ser contornado, mas a propriedade resultante é muito mais complicada. Por esse motivo, a alternativa mais simples proposta por Aguilera et al. [2000] é o preferido.

Nota 4. Falhas bizantinas e contaminação. A contaminação não pode ser evitada diante de falhas arbitrárias. Isso ocorre porque um processo defeituoso pode ser inconsistente, mesmo que entregue todas as mensagens corretamente. Ele pode então contaminar os outros processos ao transmitir uma mensagem falsa que parece correta para todos os outros processos [Hadzilacos e Toueg 1994].

2.6. Outras propriedades de pedidos

A propriedade Total Order (ver Seção 2.3), restringe a ordem de entrega das mensagens com base apenas nos destinos, ou seja, a propriedade é independente dos processos remetentes. A definição pode ser ainda mais restrita por duas propriedades relacionadas aos remetentes, a saber, *Ordem FIFO* e *Ordem Causal.*

2.6.1. Ordem FIFO. O Total Order por si só não garante que as mensagens sejam entregues na ordem em que são enviadas (ou seja, primeiro a entrar/primeiro a sair). No entanto, essa propriedade às vezes é exigida por aplicativos além do Pedido Total. A propriedade é chamada FIFO Order:

(FIFO ORDER) Se um processo correto TO transmite uma mensagem *m* antes que ele TO transmita uma mensagem *m',* então nenhum processo correto entrega *m ',* a menos que tenha anteriormente entregue m.

2.6.2. Ordem causal. A noção de causalidade no contexto de sistemas distribuídos foi formalizada pela primeira vez por Lamport [1978b]. Baseia-se na relação “precede”[3] (denotada por ÿÿ), definida em seu artigo seminal e estendida em um artigo posterior [Lamport 1986b]. A relação “precede” é definida da seguinte forma.

Definição 1. Sejam *ei* e *ej* dois eventos em um sistema distribuído. A relação transitiva *ei* ÿÿ *e* j é válida se qualquer uma das três condições a seguir for satisfeita:

(1) *ei* e *ej* são dois eventos no mesmo processo, e *ei vem* antes *de ej ;* (2) *ei* é o envio de uma mensagem *m* por um processo, e *ej* é o recebimento de *m* por outro processo; ou,

(3) Existe um terceiro evento *ek,* tal que, *ei* ÿÿ *ek* e *ek* ÿÿ *e j* (transitividade).

Essa relação define uma ordenação parcial irreflexiva no conjunto de eventos. A causalidade das mensagens pode ser definida pela relação “anterior” entre seus respectivos eventos de envio. Mais precisamente, diz-se que uma mensagem *m* precede uma mensagem *m* (denotada *m* ÿ *m),* se o evento de envio de *m* precede o evento de envio de *m .*

A propriedade de ordem causal para mensagens de difusão é definida da seguinte forma [Hadzilacos e Toueg 1994]:

(ORDEM CAUSAL) Se a transmissão de uma mensagem *m* precede causalmente a transmissão de uma mensagem *m ',* então nenhum processo correto entrega *m* a menos que tenha enviado anteriormente m.

Hadzilacos e Toueg [1994] também provam que a propriedade de Ordem Causal é equivalente a combinar a propriedade de Ordem FIFO com a seguinte propriedade de Ordem Local.

(LOCAL ORDER) Se um processo transmite uma mensagem *m* e um processo entrega *m* antes de transmitir *m',* então nenhum processo correto entrega *m* ously m . , a menos que tenha previ

Nota 5. Abordagem da máquina de estados. Uma transmissão de ordem total garantindo a ordem causal

[3] Lamport inicialmente chamou a relação de “aconteceu antes” [Lamport 1978b], mas a renomeou como “precede” em trabalhos posteriores [Lamport 1986a, 1986b].

é, por exemplo, exigido pela abordagem da máquina de estado [Lamport 1978a; Schneider 1990]. No entanto, pensamos que algumas aplicações podem exigir causalidade, outras não.

2.6.3. Ordenação da Fonte. Alguns artigos (por exemplo, Garcia-Molina e Spauster [1991] e Jia [1995]) fazem uma distinção entre a ordenação de fonte única e de fonte múltipla. Esses artigos definem algoritmos de ordenação de fonte única como algoritmos que garantem a ordem total apenas se um *único* processo transmitir mensagens. Este é um caso especial de transmissão FIFO, facilmente resolvido usando números de sequência. A ordem de origem não é particularmente interessante por si só e, portanto, não discutiremos o assunto mais adiante neste artigo.

3. PROPRIEDADES DOS GRUPOS DE DESTINO

Até agora, apresentamos o problema do broadcast de ordem total, no qual as mensagens são enviadas para todos os processos do sistema. Em outras palavras, todas as mensagens válidas são enviadas para todo o sistema:

$$\ddot{y}m \ddot{y} M \ (Dest(m) =). \quad (3)$$

Uma primitiva *multicast* é mais geral no sentido de que pode enviar mensagens para qualquer subconjunto escolhido dos processos. Em outras palavras, podemos ter duas mensagens válidas enviadas para conjuntos de destinos diferentes, ou o conjunto de destinos pode não incluir o remetente da mensagem:

$$\ddot{y}m \ddot{y} M(remetente(m) \ddot{y} \text{Dest(m)}) \ddot{y}$$
$$\ddot{y}mi, mj \ddot{y} M(Dest(mi) = Dest(mj)). \quad (4)$$

Embora amplamente utilizada, a distinção entre broadcast e multicast não é suficientemente precisa. Isso nos leva a discutir uma distinção mais relevante, a saber, entre grupos fechados versus grupos abertos e entre grupos únicos versus grupos múltiplos.

3.1. Grupos fechados versus grupos abertos

Na literatura, muitos algoritmos são projetados com a suposição implícita de que as mensagens são enviadas *dentro de* um grupo de processos. Isso veio originalmente do fato de que os primeiros trabalhos sobre esse tópico foram feitos no contexto de máquinas paralelas [Lamport 1978a] ou sistemas de armazenamento altamente disponíveis [Cristian et al. 1995]. No entanto, a maioria dos aplicativos distribuídos agora é desenvolvida considerando modelos de interação mais abertos, como o modelo cliente-servidor, arquiteturas de N camadas ou publicação/assinatura. Por esta razão, é necessário que um processo seja capaz de enviar mensagens multicast para um grupo ao qual não pertença. Conseqüentemente, consideramos uma característica importante dos algoritmos serem facilmente adaptáveis a modelos de interação aberta.

3.1.1. Algoritmos de Grupo Fechado. Em algoritmos de grupos fechados, o processo de envio é sempre um dos processos de destino:

$$\ddot{y}m \ddot{y} M \ (remetente(m) \ddot{y} Dest(m)). \quad (5)$$

Portanto, esses algoritmos não permitem que processos externos (processos que não sejam membros do grupo) enviem mensagens multicast para o grupo de destino.

3.1.2. Algoritmos de grupo aberto. Por outro lado, os algoritmos de grupo aberto permitem que qualquer processo arbitrário no sistema envie mensagens multicast para um grupo, quer o processo emissor pertença ou não ao grupo de destino. Mais precisamente, existem algumas mensagens válidas em que o remetente não é um dos destinos:

$$\ddot{y}m \ddot{y} M \ (remetente(m) \ddot{y} \text{Dest(m)}). \quad (6)$$

Os algoritmos de grupo aberto são mais gerais do que os algoritmos de grupo fechado: o primeiro pode ser usado com grupos fechados, enquanto o oposto não é verdadeiro.

3.2. Grupos Únicos Versus Múltiplos

A maioria dos algoritmos apresentados na literatura assume que todas as mensagens são multicast para um único grupo de processos de destino. No entanto, alguns algoritmos são projetados para suportar vários grupos. Neste contexto, consideramos três situações: *grupo único, grupos múltiplos disjuntos* e

vários grupos sobrepostos. Também discutimos como soluções inúteis e triviais podem ser descartadas com a noção de *minimalidade.*
Uma vez que a capacidade de multicast de mensagens para vários conjuntos de destino é crítica para certas classes de aplicativos, consideramos essa capacidade como uma característica importante de um algoritmo.

3.2.1. Ordenação de Grupo Único. Com a ordenação de grupo único, todas as mensagens são multicast para um único grupo de processos de destino. Como mencionado acima, este é o modelo considerado por uma grande maioria dos algoritmos que são estudados neste artigo.
A ordem de um único grupo pode ser definida pela seguinte propriedade:4

ÿmi, mj ÿ *M(Dest(mi) = Dest(mj)).* (7)

3.2.2. Ordenação de Grupos Múltiplos (Disjuntos). Em alguns aplicativos, a restrição a um único grupo de destino não é aceitável.
Por esta razão, foram propostos algoritmos que suportam mensagens multicast para vários grupos.
O caso mais simples ocorre quando os grupos múltiplos são grupos *disjuntos .* Mais precisamente, se duas mensagens válidas tiverem conjuntos de destino diferentes, então esses conjuntos não se cruzam:

ÿmi, mj ÿ *M(Dest(mi)* = Dest(mj) ÿ Dest(mi) ÿ Dest(mj) = ÿ). (8)

Adaptar algoritmos projetados para um único grupo funcionar em um sistema com vários grupos disjuntos é quase trivial.

3.2.3. Ordenação de vários grupos (ping de sobreposição). No caso de ordenação de vários grupos, pode acontecer que os grupos se sobreponham. Isso pode ser expresso pelo fato de que alguns pares de mensagens válidas têm conjuntos de destino diferentes com um destino não vazio.

4Esta definição e as seguintes são estáticas.
Eles não levam em conta o fato de que os processos podem entrar e sair de grupos. No entanto, preferimos essas definições estáticas simples, em vez de definições mais complexas que levariam em conta grupos de destino dinâmicos.

interseção:

ÿmi, mj ÿ *M(Dest(mi)* = Dest(mj) ÿ Dest(mi) ÿ Dest(mj) = ÿ). (9)

A dificuldade real de projetar algoritmos multicast de ordem total para vários grupos surge quando os grupos podem se sobrepor. Isso é facilmente compreendido quando se considera o problema de garantir total ou der na intersecção de grupos.
Nesse contexto, Hadzilacos e Toueg [1994] fornecem três propriedades diferentes para a ordem total na presença de múltiplos grupos: *Ordem Total Local, Ordem Total Pairwise* e *Ordem Total Global.* 5

(LOCAL TOTAL ORDER) Se os processos corretos *p* e *q ,* ambos TO-entregam as mensagens *m* e *m* e *Dest(m) = Dest(m),* então *p* TO-entrega *m* antes de *m* se e somente se *q* TO-entrega *m* antes de *m* .

Ordem total local é a mais fraca das três propriedades. Ele requer que total ou der seja aplicado apenas para mensagens multicast dentro do mesmo grupo.
Observe também que vários grupos não relacionados podem ser considerados grupos separados, mesmo que se sobreponham. De fato, os processos de destino pertencentes à interseção de dois grupos podem ser vistos como tendo duas identidades distintas, uma para cada grupo.
Segue-se que um algoritmo para grupos múltiplos distintos pode ser adaptado trivialmente para suportar grupos sobrepostos com Ordem Total Local.

Conforme apontado por Hadzilacos e Toueg [1994], o primitivo multicast de ordem total da primeira versão do Isis [Birman e Joseph 1987] garantiu a Ordem Total Local.6

(PAIRWISE TOTAL ORDER) Se dois processos corretos, *p* e *q ,* ambos TO-entregam mensagens *m* e *m ,* então *p* TO-entrega *m*

5As propriedades de ordenação citadas aqui estão sujeitas a contaminação, consulte a Seção 2.5. A contaminação pode ser evitada formulando essas propriedades de maneira semelhante à propriedade Ordem Total Uniforme sem Intervalos.
6De notar que, se a transformação é trivial do ponto de vista conceptual, a implementação foi certamente uma questão totalmente diferente, sobretudo em meados dos anos 80.

antes de *m* , se e somente se *q* TO-entrega *m* antes de *m* .

O pedido total em pares é estritamente mais forte que o pedido total local. Mais notavelmente, requer que a ordem total seja aplicada a todas as mensagens entregues na interseção de dois grupos.

Até onde sabemos, não existe um algoritmo direto para transformar um algoritmo multicast de ordem total que imponha a Ordem Total Local em um que também garanta a Ordem Total Pairwise (exceto para soluções triviais; consulte a Seção 3.2.4).

Hadzilacos e Toueg [1994] observam que, por exemplo, Pairwise Total Order é a propriedade de ordem garantida pelo algoritmo de Garcia-Molina e Spauster [1989, 1991].

A ordem total de pares por si só pode levar a situações inesperadas quando há três ou mais grupos de destino sobrepostos. Por exemplo, Fekete [1993] ilustra o problema com o seguinte cenário. Considere três processos *pi, pj , pk* e três mensagens m1, m2, m3 que são enviadas respectivamente para três grupos diferentes de sobreposição G1 = *{pi,* pj}, G2 = *{pj ,* pk} e G3 = *{pk,* pi }. Pairwise Total Order permite os seguintes históricos em *pi, pj , pk:*

pi : ⋯*TO-deliver(m3)* ÿÿ · · · ÿÿ *TO-deliver(m1)* ⋯*pj :* ⋯*TO-deliver(m1)* ÿÿ · · · ÿÿ *TO-deliver(m2)* ⋯*pk :* ⋯*TO-deliver(m2)* ÿÿ · · · ÿÿ *TO-deliver(m3)* ⋯

Esta situação é evitada pela especificação da Ordem Total Global [Hadzilacos e Toueg 1994], que é definida da seguinte forma:

(GLOBAL TOTAL ORDER) A relação < é acíclica, onde < é definido da seguinte forma: *m* < *m* se e somente se algum processo correto entregar *m* e *m* , naquela ordem.

Nota 6. Fekete [1993] fornece outra especificação para multicast de ordem total que também evita o cenário mencionado. A especificação, chamada AMC, é expresso como um autômato de E/S [Lynch e Tuttle 1989; Lynch 1996] e usa a noção de pseudo-tempo para impor uma ordem na entrega de mensagens.

3.2.4. Minimalidade e soluções triviais. Qualquer algoritmo que resolva o problema de transmissão de ordem total em um único grupo pode ser facilmente adaptado para resolver o problema de vários grupos com a seguinte abordagem:

(1) formar um supergrupo com a união de todos os grupos de destino; (2) sempre que uma mensagem *m* for multicast para um grupo, multicast para o supergrupo, e

(3) processos que não estão em *Dest(m)* descartam m.

O problema com essa abordagem é sua inerente falta de escalabilidade. De fato, em sistemas distribuídos muito grandes, mesmo que os grupos de destino sejam individualmente pequenos, é provável que sua união cubra um número muito grande de processos.

Para evitar esse tipo de solução, Guerraoui e Schiper [2001] exigem a implementação de multicast de ordem total para vários grupos para satisfazer a seguinte propriedade de minimalidade:

(MINIMALIDADE FORTE) A execução do algoritmo implementando multicast de ordem total para uma mensagem *m* envolve apenas *sender(m)* e os processos em Dest(m).

Essa propriedade costuma ser forte demais: ela impede muitos algoritmos interessantes que usam um pequeno número de processos externos para ordenação de mensagens (por exemplo, algoritmos que disseminam mensagens ao longo de alguma árvore de propagação). Uma propriedade mais fraca permitiria que um algoritmo envolvesse um pequeno conjunto de processos externos.

3.2.5. Algoritmo de Transformação. Delporte Gallet e Fauconnier [2000] propõem um algoritmo genérico que transforma um algoritmo de broadcast de ordem total para um único grupo fechado em um para múltiplos grupos.
O algoritmo divide os grupos de destino em entidades menores e suporta vários grupos com Minimalidade Forte.

3.3. Grupos Dinâmicos

A especificação da Seção 2 é a especificação padrão da ordem total transmitida em um sistema *estático ,* isto é, um sistema no qual todos os processos são criados na inicialização do sistema. Na prática, no entanto, é desejável que os processos entrem e saiam dos grupos em tempo de execução.

Um grupo *dinâmico* é um grupo de processos com membros que podem mudar durante a computação: os processos podem entrar ou sair dinamicamente do grupo, ou podem ser removidos do grupo (a remoção em face de falhas é discutida posteriormente na Seção 6.2). . Com um grupo dinâmico, os membros sucessivos do grupo são chamados de *visões* do grupo [Chockler, Keidar e Vitenberg 2001].

Com grupos dinâmicos, a abstração de comunicação básica é chamada de *sincronia de visualização,* que pode ser vista como a contrapartida de transmissão confiável em sistemas estáticos. A transmissão confiável é definida pelas propriedades Validade, Acordo e Integridade Uniforme da Seção 2. Grosso modo, a sincronia de exibição adota uma definição semelhante, enquanto relaxa a propriedade Acordo.7 A transmissão de ordem total em um sistema com grupos dinâmicos pode, portanto, ser especificado como sincronia de visualização, mais uma propriedade de ordem total.

3.4. Grupos Particionáveis

Em uma rede de longa distância, a rede pode se tornar temporariamente particionada; ou seja, alguns dos nós não podem mais se comunicar, pois todos os links entre eles estão quebrados. Quando isso acontece, os grupos de destino podem ser divididos em vários subgrupos isolados (ou partições). Existem duas abordagens principais para lidar com grupos particionados: (1) a associação à *partição primária* e (2) a associação *particionável .*

Com a associação de partição primária, uma das partições é reconhecida como a partição primária.8 Somente processos que pertencem à partição primária têm permissão para entregar mensagens, enquanto os outros processos devem esperar até que possam mesclar novamente com a partição primária.

Em contraste, a associação ao grupo particionável permite que todos os processos entreguem mensagens, independentemente da partição a que pertencem. Fazer isso requer adaptar a especificação da transmissão de ordem total. Chockler, Keidar e Vitenberg [2001] definem três propriedades de ordem em um sistema particionável: Forte Ordem Total (as mensagens são entregues na mesma ordem por todos os processos que as entregam), Fraca Ordem Total (o requisito da ordem é restrito em uma exibição) e Ordem total confiável (estende a propriedade Ordem total forte para exigir que os processos entreguem um prefixo de uma sequência comum de mensagens em cada exibição). Em outras palavras, com apenas pequenas diferenças, Ordem Total Forte corresponde à propriedade Ordem Total Uniforme da Seção 2.3, e Ordem Total Confiável à propriedade Ordenação por Prefixo da Seção 2.5. Outras propriedades, como Validade, também são definidas de forma diferente em sistemas particionáveis. Isso é explicado com mais detalhes por Chockler, Kei dar e Vitenberg [2001] e Fekete et al. [2001].

4. MECANISMOS DE PEDIDO DE MENSAGEM

Nesta seção, propomos uma classificação de algoritmos de broadcast de ordem total na ausência de falhas. A primeira pergunta que fazemos é: “quem constrói a ordem?” Mais especificamente, estamos interessados na entidade que gera as informações necessárias para definir a ordem das mensagens (por exemplo, timestamp ou número de sequência).

Identificamos três papéis diferentes que um processo participante pode assumir em relação ao algoritmo: remetente, destino ou sequenciador. Um processo *remetente* é um processo *ps* do qual uma mensagem se origina (ou seja, *ps* ÿ *remetente).* Um processo *de destino* é

7Discutir esse primitivo em detalhes está além do escopo desta pesquisa (consulte o artigo de Chockler, Keidar e Vitenberg [2001] para obter detalhes).

8Uma maneira simples de fazer isso é reconhecer como primário partição apenas uma que retém a maioria dos processos da visão anterior. Isso não garante que sempre exista uma partição primária, mas garante que, se existir, ela será única.

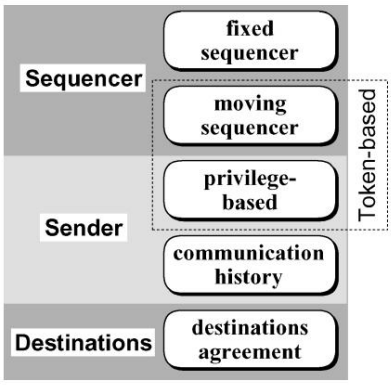


Fig. 3. Classes de algoritmos de broadcast de ordem total.

um processo *pd* para o qual uma mensagem é enviada (ou seja, *pd* ÿ *dest).* Finalmente, um processo *sequenciador* não é necessariamente um remetente ou um destino, mas está de alguma forma envolvido na ordenação das mensagens. Um determinado processo pode assumir simultaneamente vários papéis (por exemplo, remetente , sequenciador e destino). No entanto, representamos esses papéis separadamente, pois são conceitualmente diferentes.

De acordo com os três diferentes papéis, definimos três classes básicas para algoritmos de broadcast total ou der, dependendo se a ordem é construída respectivamente por um sequenciador, pelo remetente ou pelos processos de destino. Entre algoritmos da mesma classe, diferenças significativas permanecem.
Para explicar esse problema, introduzimos uma divisão adicional, levando a cinco subclasses no total. Essas classes são nomeadas da seguinte forma (veja a Figura 3): *sequenciador fixo, sequenciador móvel, baseado em privilégios, histórico de comunicação* e *acordo de destinos.* Algoritmos baseados em privilégios e sequenciadores móveis são comumente referidos como algoritmos baseados em tokens.

A terminologia definida neste artigo é parcialmente emprestada de outros autores. Por exemplo, "história da comunicação" e "sequenciador fixo" foram propostos por Cristian e Mishra [1995]. O termo "baseado em privilégios" foi sugerido por Dahlia Malkhi em uma discussão privada. Finalmente, Le Lann e Bres [1991] agrupam os algoritmos em três classes, com base no local onde a ordem é construída. Infelizmente, sua definição

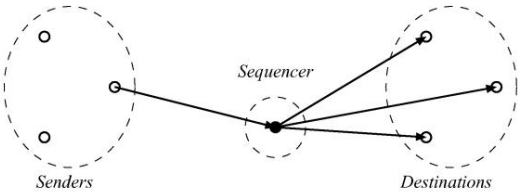


Fig. 4. Algoritmos de sequenciador fixo.

de classes é específico para uma arquitetura cliente-servidor.

No restante desta seção, apresentamos cada uma das cinco classes e ilustramos cada classe com um algoritmo simples. Os algoritmos são meramente apresentados com a finalidade de ilustrar a categoria correspondente e não devem ser considerados como exemplos de trabalho completos. Embora inspirados em algoritmos existentes, eles são amplamente simplificados e nenhum deles é tolerante a falhas.

Nota 7. *Blocos atômicos.* Os algoritmos são escritos em pseudocódigo, assumindo que os blocos associados a uma cláusula when são executados atomicamente em relação às cláusulas when do mesmo processo, exceto quando um processo é bloqueado em uma instrução wait. Essa suposição simplifica bastante a expressão dos algoritmos com relação à simultaneidade.

4.1. Sequenciador fixo

Em um algoritmo de sequenciador fixo, um processo é eleito como o sequenciador e é responsável por ordenar as mensagens. O sequenciador é único, e a responsabilidade normalmente não é transferida para outros processos (pelo menos na ausência de falha).

A abordagem é ilustrada na Figura 4 e na Figura 5. Um processo específico assume o papel de um sequenciador e constrói a ordem total. Para difundir uma mensagem m, um remetente envia *m* para o sequenciador. Ao receber m, o seqüenciador atribui a ele um número de sequência e retransmite *m* com seu número de sequência para os destinos. O último, então, entrega mensagens de acordo com os números de sequência. Este algoritmo não tolera a falha do sequenciador.

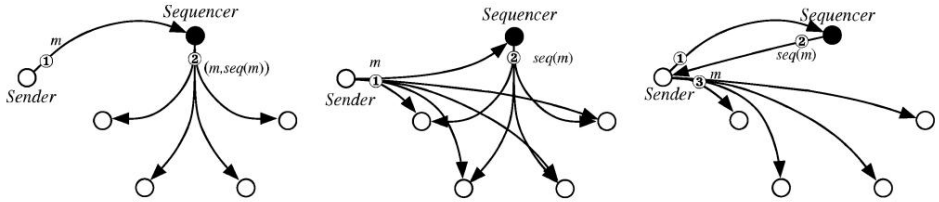
De fato, existem três variantes de algoritmos de sequenciador fixo. Chamamos essas três variantes de "UB" (unicast-broadcast),

```
Sender:
    procedure TO-broadcast(m)                    { To TO-broadcast a message m }
        send (m) to sequencer

Sequencer:
    Initialization:
        seqnum := 1
    when receive (m)
        sn(m) := seqnum
        send (m, sn(m)) to all
        seqnum := seqnum + 1

Destinations (code of process p_i):
    Initialization:
        nextdeliver_{p_i} := 1
        pending_{p_i} := ∅
    when receive (m, seqnum)
        pending_{p_i} := pending_{p_i} ∪ {(m, seqnum)}
        while ∃(m', seqnum') ∈ pending_{p_i} : seqnum' = nextdeliver_{p_i} do
            deliver (m')
            nextdeliver_{p_i} := nextdeliver_{p_i} + 1
```

Fig. 5. Algoritmo de sequenciador fixo simples.



(a) variante UB (b) variante BB (c) variante UUB

Fig. 6. Variantes comuns de algoritmos de sequenciador fixo.

"BB" (broadcast-broadcast) e "UUB" (unicast-unicast-broadcast), inspirando-se em Kaashoek e Tanenbaum [1996].

Na primeira variante, denominada "UB" (ver Figura 6(a)), o protocolo consiste em um unicast para o sequenciador, seguido por um broad cast do sequenciador. Essa variante gera poucas mensagens e é a mais simples das três abordagens. É, por exemplo, adotado por Navaratnam et al. [1988], e corresponde ao algoritmo da Figura 5.

Na segunda variante, denominada "BB" (Figura 6(b)), o protocolo consiste em um broadcast para todos os destinos mais o sequenciador, seguido por um segundo broadcast do sequenciador. Isso gera mais mensagens do que a abordagem anterior, exceto em redes de transmissão. No entanto, pode reduzir a carga no sequenciador e torna mais fácil tolerar a falha do sequenciador. Isis (sequenciador) [Birman et al. 1991] é um exemplo da segunda variante.

A terceira variante, chamada "UUB" (Figura 6(c)), é menos comum que as demais. Em suma, o protocolo consiste nas seguintes etapas. O remetente solicita um número de sequência do sequenciador (unicast). O sequenciador responde com um número de sequência (unicast). Em seguida, o remetente transmite a mensagem sequenciada para os processos de destino.9

9O protocolo para tolerar falhas é complexo.

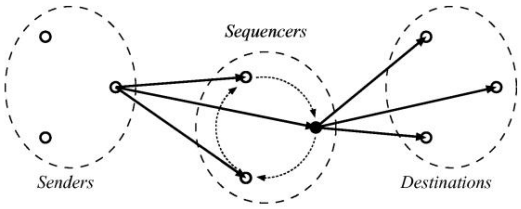


Fig. 7. Algoritmos do sequenciador móvel.

4.2. Sequenciador de movimento

Os algoritmos do sequenciador móvel são baseados no mesmo princípio dos algoritmos do sequenciador fixo, mas permitem que o papel do sequenciador seja transferido entre vários processos. A motivação é distribuir a carga entre eles. Isso é ilustrado na Figura 7, onde o sequenciador é escolhido entre vários processos. O código executado por cada processo é, no entanto, mais complexo do que com um sequenciador fixo, o que explica a popularidade da última abordagem. Observe que, com algoritmos de sequenciador móvel, as funções do sequenciador e dos processos de destino são normalmente combinadas.

A Figura 8 mostra o princípio dos algoritmos do sequenciador móvel. Para difundir uma mensagem m, um remetente envia *m* para os sequenciadores. Os sequenciadores circulam uma mensagem token que carrega um número de sequência e uma lista de todas as mensagens às quais um número de sequência foi atribuído (isto é, todas as mensagens sequenciadas). Ao receber o token, um sequenciador atribui um número de sequência a todas as mensagens recebidas, mas ainda não sequenciadas. Ele envia as novas mensagens sequenciadas para os destinos, atualiza o token e o passa para o próximo sequenciador.

Nota 8. Semelhante aos algoritmos de sequenciador fixo, é possível desenvolver um algoritmo de sequenciador móvel de acordo com uma das três variantes. No entanto, a diferença entre as variantes não é tão nítida quanto para um sequenciador fixo. Acontece que todos os algoritmos do sequenciador móvel pesquisados seguem o equivalente da variante do sequenciador fixo BB. Portanto, não discutiremos mais essa questão.

Nota 9. Conforme mencionado, a principal motivação para usar um sequenciador móvel é distribuir a carga entre vários processos, evitando assim o gargalo causado por um único processo. Isso é ilustrado em vários estudos (por exemplo, Cristian et al. [1994] e Urban et al. [2000]). Alguém poderia então se perguntar por que um algoritmo de sequenciador fixo deve ser preferido a um algoritmo de sequenciador móvel. Há, de fato, pelo menos três razões possíveis. Primeiro, os algoritmos do sequenciador fixo são consideravelmente mais simples, deixando menos espaço para erros de implementação. Em segundo lugar, a latência de algoritmos sequenciadores fixos geralmente é melhor, como mostrado por Urban ´ et al. [2000]. Em terceiro lugar, muitas vezes algumas máquinas são mais confiáveis, mais confiáveis, mais bem conectadas ou simplesmente mais rápidas do que outras. Nesse caso, faz sentido usar um deles como sequenciador fixo (consulte M

4.3. Baseado em privilégio

Algoritmos baseados em privilégios dependem da ideia de que os remetentes podem transmitir mensagens apenas quando recebem o privilégio de fazê-lo. A Figura 9 ilustra esta classe de algoritmos. A ordem é definida pelos remetentes quando eles transmitem suas mensagens. O privilégio de transmitir (e ordenar) mensagens é concedido a apenas um processo por vez, mas esse privilégio circula de processo para processo, entre os remetentes. Em outras palavras, devido à arbitragem entre os remetentes, construir a ordem total requer resolver o problema do broadcast FIFO (facilmente resolvido com números de sequência no remetente) e garantir que a passagem do privilégio para o próximo remetente não viole essa ordem.

A Figura 10 ilustra o princípio dos algoritmos baseados em privilégios. Os remetentes circulam uma mensagem de token que carrega um número de sequência a ser usado ao transmitir a próxima mensagem. Quando um processo deseja transmitir uma mensagem m, ele deve primeiro esperar até receber a mensagem to ken. Em seguida, atribui um número de sequência a cada uma de suas mensagens e as envia a todos os destinos. Em seguida, o remetente atualiza o token e o envia para o próximo remetente. Os processos de destino entregam mensagens em números de sequência crescentes.

```
Sender:
    procedure TO-broadcast(m)                    { To TO-broadcast a message m }
        send (m) to all sequencers

Sequencers (code of process s_i):
    Initialization:
        received_{s_i} := ∅
        if s_i = s_1 then
            token.seqnum := 1
            token.sequenced := ∅
            send token to s_1
    when receive m
        received_{s_i} := received_{s_i} ∪ {m}
    when receive token from s_{i-1}
        for each m' in received_{s_i} \ token.sequenced do
            send (m', token.seqnum) to destinations
            token.seqnum := token.seqnum + 1
            token.sequenced := token.sequenced ∪ {m'}
        send token to s_{i+1 (mod n)}

Destinations (code of process p_i):
    Initialization:
        nextdeliver_{p_i} := 1
        pending_{p_i} := ∅
    when receive (m, seqnum)
        pending_{p_i} := pending_{p_i} ∪ {(m, seqnum)}
        while ∃(m', seqnum') ∈ pending_{p_i} s.t. seqnum' = nextdeliver_{p_i} do
            deliver (m')
            nextdeliver_{p_i} := nextdeliver_{p_i} + 1
```

Fig. 8. Algoritmo de sequenciador de movimento simples.

Nota 10. Em algoritmos baseados em privilégios, os remetentes geralmente precisam se conhecer para distribuir o privilégio. Essa restrição torna os algoritmos baseados em privilégios pouco adequados para grupos abertos, onde não há um conjunto fixo e previamente conhecido de remetentes.

Nota 11. Em sistemas síncronos, os algoritmos baseados em privilégios baseiam-se na ideia de que cada processo emissor pode enviar mensagens apenas durante intervalos de tempo predeterminados. Esses intervalos de tempo são atribuídos a cada processo de forma que dois processos não possam enviar mensagens ao mesmo tempo. Ao garantir que o meio de comunicação seja acessado em exclusão mútua, a ordem total é facilmente garantida. A técnica também é conhecida como *acesso múltiplo por divisão de tempo* (TDMA).

Nota 12. É tentador considerar que os algoritmos do sequenciador baseado em privilégios e em movimento são equivalentes, pois ambos dependem em um mecanismo de passagem de token. No entanto, eles diferem em um aspecto significativo: a ordem total é construída por remetentes em algoritmos baseados em privilégios, enquanto é construída por sequenciadores em algoritmos de sequenciadores móveis. Isso tem pelo menos duas consequências importantes. Primeiro, os algoritmos do sequenciador móvel são facilmente adaptados a grupos abertos. Em segundo lugar, em algoritmos baseados em privilégios, a passagem do token é necessária para garantir a vivacidade do algoritmo, enquanto nos algoritmos de sequenciador móvel, ele é usado principalmente para

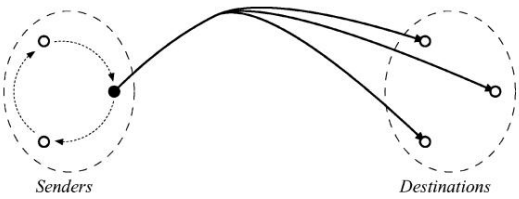


Fig. 9. Algoritmos baseados em privilégios.

```
Senders (code of process s_i):
    Initialization:
        tosend_{s_i} := ∅
        if s_i = s_1 then
            token.seqnum := 1
            send token to s_1
    procedure TO-broadcast(m)                    { To TO-broadcast a message m }
        tosend_{s_i} := tosend_{s_i} ∪ {m}
    when receive token
        for each m' in tosend_{s_i} do
            send (m', token.seqnum) to destinations
            token.seqnum := token.seqnum + 1
        tosend_{s_i} := ∅
        send token to s_{i+1 (mod n)}

Destinations (code of process p_i):
    Initialization:
        nextdeliver_{p_i} := 1
        pending_{p_i} := ∅
    when receive (m, seqnum)
        pending_{p_i} := pending_{p_i} ∪ {(m, seqnum)}
        while ∃(m', seqnum') ∈ pending_{p_i} s.t. seqnum' = nextdeliver_{p_i} do
            deliver (m')
            nextdeliver_{p_i} := nextdeliver_{p_i} + 1
```

Fig. 10. Algoritmo simples baseado em privilégios.

melhorando o desempenho, por exemplo, balanceamento de carga.

Nota 13. É difícil garantir a imparcialidade com algoritmos baseados em privilégios. De fato, se um processo tiver um número muito grande de mensagens para transmitir, ele poderá manter o token por um tempo arbitrariamente longo, evitando assim que outros processos transmitam suas próprias mensagens. Para superar esse problema, os algoritmos geralmente impõem um limite superior no número de mensagens e/ou no tempo que algum processo pode manter o token. Uma vez ultrapassado o limite, o processo é obrigado a liberar o token, independentemente do número de mensagens restantes a serem transmitidas.

4.4. História da Comunicação

Em algoritmos de histórico de comunicação, como em algoritmos baseados em privilégios, a ordem de entrega é determinada pelos remetentes. No entanto, em contraste com os algoritmos baseados em privilégios, os processos podem transmitir mensagens a qualquer momento e a ordem total é garantida pelo atraso na entrega das mensagens. As mensagens geralmente carregam um (físico ou lógico) carimbo de data/hora. Os destinos observam as mensagens geradas pelos demais processos e seus timestamps, ou seja, o histórico de comunicação no sistema, para saber quando a entrega de uma mensagem não irá mais violar a ordem total.

Existem duas variantes fundamentalmente diferentes de algoritmos de histórico de comunicação. Na primeira variante, denominada *história causal,* os algoritmos da história da comunicação utilizam uma ordem parcial, definida pela história causal das mensagens, e transformam essa ordem parcial em uma ordem total.
As mensagens simultâneas são ordenadas de acordo com alguma função predeterminada. Na segunda variante, conhecida como *mesclagem determinística,* os processos enviam mensagens com tempos tamponados de forma independente (não refletindo assim a ordem causal), e a entrega ocorre de acordo com uma política determinística de mesclar os fluxos de mensagens provenientes de cada processo.

A Figura 11 ilustra um algoritmo de histórico de comunicação típico da primeira variante. O algoritmo, inspirado em Lamport [1978b], funciona da seguinte maneira. O algoritmo usa relógios lógicos [Lamport 1978b] para

```
Senders and destinations (code of process p; assumes FIFO channels):
    Initialization:
        received_p := ∅                                   { Messages received by process p }
        delivered_p := ∅                                  { Messages delivered by process p }
        LC_p[p_1 ... p_n] := {0, ..., 0}    { LC_p[q]: logical clock of process q, as seen by p }
    procedure TO-multicast(m)                             { To TO-multicast a message m }
        LC_p[p] := LC_p[p] + 1
        ts(m) := LC_p[p]
        send FIFO (m, ts(m)) to all
    when receive (m, ts(m))
        LC_p[p] := max(ts(m), LC_p[p]) + 1
        LC_p[sender(m)] := ts(m)
        received_p := received_p ∪ {m}
        deliverable := ∅
        for each message m' in received_p \ delivered_p do
            if ts(m') ≤ min_{q∈Π} LC_p[q] then
                deliverable := deliverable ∪ {m'}
        deliver all messages in deliverable, in increasing order of (ts(m), sender(m))
        delivered_p := delivered_p ∪ deliverable
```

Fig. 11. Algoritmo de histórico de comunicação simples (histórico causal).

```
Senders and destinations (assumes FIFO channels):
    procedure TO-multicast(m)                             { To TO-multicast a message m }
        send FIFO (m) to all
    forever do
        for each process p in Π_sender do
            wait until receive m' from p
            deliver (m')
```

Fig. 12. Algoritmo de histórico de comunicação simples (fusão determinística).

"timestamp" cada mensagem *m* com o tempo lógico do evento *TO-broadcast(m) ,* denotado *ts(m).* As mensagens são então entregues na ordem de seus timestamps. No entanto, podemos ter duas mensagens, *m* e *m* cọm o mesmo timestamp. Para arbitrar entre essas mensagens, o algoritmo utiliza a ordem lexicográfica nos identificadores dos processos remetentes. Na Figura 11, nos referimos a essa ordem como (ts(m), *sender(m))* ou der, onde *sender(m)* é o identificador do processo remetente.

Um exemplo simples da segunda variante é ilustrado na Figura 12. O algoritmo assume que a comunicação é FIFO e que o remetente processa mensagens de broadcast na mesma taxa. Os processos de destino executam um loop infinito onde aceitam, de forma round-robin, uma única mensagem de cada processo emissor. Aguilera e Strom [2000] (Seção 7.4.9), por exemplo, propõem um algoritmo mais elaborado baseado no mesmo princípio.

Nota 14. Os algoritmos da Figura 11 e da Figura 12 não são ativos. De fato, considere o algoritmo da Figura 11 e um cenário em que um único processo *p* transmite uma única mensagem m, enquanto nenhum outro processo jamais transmite qualquer mensagem. De acordo com o algoritmo da Figura 11, um processo *q* pode entregar *m* somente após ter recebido, de todos os processos, uma mensagem que foi transmitida *após* a recepção de m. Isso é, obviamente, impossível se pelo menos um dos processos nunca transmitir qualquer mensagem. Para contornar esse problema, os algoritmos de histórico de comunicação propostos na literatura geralmente enviam mensagens vazias quando nenhuma mensagem do aplicativo é transmitida.

Nota 15. Em sistemas síncronos, os algoritmos de histórico de comunicação dependem de relógios sincronizados, e usam timestamps físicos (timestamps provenientes dos relógios sincronizados), em vez de lógicos. A natureza de tais sistemas torna

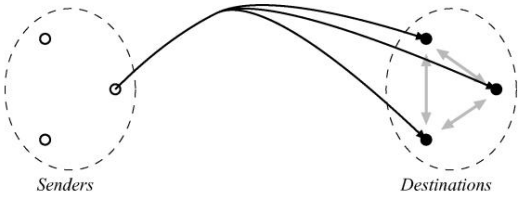


Fig. 13. Algoritmos de acordo de destinos.

desnecessário enviar mensagens vazias para garantir vivacidade. Na verdade, isso pode ser visto como um exemplo do uso do tempo para se comunicar [Lamport 1984].

4.5. Contrato de destinos

Nos algoritmos de acordo de destinos, como o nome indica, a ordem de entrega resulta de um acordo entre os processos de destino (ver Figura 13). Distinguimos três variantes diferentes de acordo: (1) acordo sobre um número de sequência de mensagem, (2) acordo sobre um conjunto de mensagens ou (3) acordo sobre a aceitação de uma ordem de mensagem proposta.

A Figura 14 ilustra um algoritmo da primeira variante: para cada mensagem, os processos de destino chegam a um acordo sobre um número de sequência único (ainda que não consecutivo). O algoritmo é adaptado do algoritmo de Skeen (Seção 7.5.1), embora opere de maneira descentralizada. Resumidamente, o algoritmo funciona da seguinte maneira. Para difundir uma mensagem m, um remetente envia *m* para todos os destinos. Ao receber m, um destino atribui a ele um timestamp local e envia esse timestamp para todos os destinos. Uma vez que um processo de destino tenha recebido um timestamp local para *m* de todos os destinos, um timestamp global único sn(m) é atribuído a m, calculado como o máximo de todos os timestamps locais. As mensagens são entregues na ordem de seu carimbo de data/hora global, ou seja, uma mensagem *m* só pode ser entregue uma vez que tenha sido atribuído seu carimbo de data/hora global *sn(m),* e nenhuma outra mensagem não entregue *m* pode possivelmente receber um carimbo de data/hora sn(m) menor ou igual a *sn(m).* Assim como no algoritmo de histórico de comunicação (Figura 11), o identificador do remetente da mensagem é usado para desempate entre mensagens com o mesmo timestamp global.

O algoritmo mais representativo da segunda variante de concordância é o algoritmo proposto por Chandra e Toueg [1996] (Seção 7.5.4). O algoritmo transforma o broadcast de ordem total em uma sequência de problemas *de consenso.10* Cada instância do consenso decide sobre um conjunto de mensagens a serem entregues, ou seja, o número de consenso *k* permite que os processos concordem com um conjunto de mensagens , *Msgk .* Para *k* < *k* as mensagens em *Msgk* são entregues antes das mensagens em *Msgk .* As mensagens em um conjunto *Msgk* são entregues de acordo com alguma ordem predeterminada (por exemplo, na ordem lexical de seus identificadores).

Com a terceira variante de acordo, uma ordem provisória de entrega de mensagem é proposta pela primeira vez (geralmente por um dos destinos). Em seguida, os processos de destino devem concordar em aceitar ou rejeitar a proposta. Em outras palavras, esta variante do acordo de destinos depende de um protocolo de compromisso atômico. O algoritmo proposto por Luan e Gligor [1990] normalmente pertence à terceira variante.

Nota 16. Há uma linha tênue entre a segunda e a terceira variantes de concordância. Por exemplo, o algoritmo de transmissão de ordem total de Chandra e Toueg [1996] depende do consenso, conforme descrito. No entanto, quando combinado com o algoritmo de consenso do coordenador rotativo [Chandra e Toueg 1996], o algoritmo resultante pode ser visto como um algoritmo da terceira forma. De fato, o coordenador propõe uma ordem provisória (dada como um conjunto de mensagem mais identificadores de mensagem) que tenta validar. Assim, é importante notar que dois algoritmos aparentemente idênticos podem usar diferentes formas de concordância, simplesmente porque são descritos em diferentes níveis de abstração.

4.6. Pedidos sem tempo versus pedidos com base no tempo

Introduzimos uma distinção adicional entre algoritmos, ortogonal ao acima

10O problema do consenso é informalmente definido da seguinte forma: todo processo propõe algum valor, e todos os processos devem eventualmente decidir sobre o mesmo valor, que deve ser um (qualquer) dos valores propostos.

```
Sender:
    procedure TO-broadcast(m)                          { To TO-broadcast a message m }
        send (m) to destinations

Destinations (code of process p_i):
    Initialization:
        stamped_{p_i} := ∅
        received_{p_i} := ∅
        LC_{p_i} := 0                                  { LC_{p_i}: logical clock of process p_i }
    when receive m
        ts_i(m) := LC_{p_i}
        received_{p_i} := received_{p_i} ∪ {(m, ts_i(m))}
        send (m, ts_i(m)) to destinations
        LC_{p_i} := LC_{p_i} + 1
    when received (m, ts_j(m)) from p_j
        LC_{p_i} := max(ts_j, LC_{p_i} + 1)
        if received (m, ts(m)) from all destinations then
            sn(m) := max_{k=1...n} ts_k(m)
            stamped_{p_i} := stamped_{p_i} ∪ {(m, sn(m))}
            received_{p_i} := received_{p_i} \ {m}
            deliverable := ∅
            for each (m', sn(m')) ∈ stamped_{p_i} s.t. ∀m'' ∈ received_{p_i} : sn(m') < ts_i(m'') do
                deliverable := deliverable ∪ {(m', sn(m'))}
            deliver all messages in deliverable in increasing order of (sn(m), sender(m))
            stamped_{p_i} := stamped_{p_i} \ deliverable
```

Fig. 14. Algoritmo de concordância de destinos simples.

classificação. A distinção é entre algoritmos que usam tempo físico para ordenação de mensagens e algoritmos que não usam tempo físico. Por exemplo, na Seção 4.4 (ver Figura 11), apresentamos um algoritmo simples de histórico de comunicação baseado em tempo *lógico* . É realmente possível projetar um algoritmo semelhante que use o tempo *físico* (e relógios sincronizados).

Resumindo, distinguimos algoritmos com *ordenação baseada no tempo,* que dependem do tempo físico, e algoritmos com *ordenação sem tempo,* que não usam o tempo físico.

5. QUESTÕES CONCEITUAIS RELACIONADAS A FALHAS

Na Seção 4, discutimos os mecanismos de pedido, ignorando o problema das falhas. Os mecanismos para tolerância a falhas são discutidos abaixo na Seção 6. No entanto, a tolerância a falhas não pode ser discutida sem alguma discussão prévia sobre questões de modelo de sistema. Isso é feito nesta seção.

5.1. Sincronia e pontualidade

A sincronia de um sistema define as suposições de tempo que são feitas sobre o comportamento de processos e canais de comunicação. Mais especificamente, costuma-se considerar dois parâmetros principais. O primeiro parâmetro é o *intervalo de velocidade do processo,* que é dado pela diferença entre a velocidade dos processos mais lentos e mais rápidos do sistema. O segundo parâmetro é o *atraso de comunicação,* que é dado pelo tempo decorrido entre o envio e o recebimento das mensagens. A sincronia do sistema é definida considerando vários limites nesses dois parâmetros.

Um sistema onde ambos os parâmetros têm um limite superior conhecido é chamado de *sistema síncrono.* No outro extremo, um sistema no qual a velocidade do processo e os atrasos de comunicação são ilimitados é chamado de *sistema assíncrono.* Entre esses dois extremos está a definição de vários modelos de sistemas parcialmente síncronos [Dolev et al. 1987; Dwork et ai. 1988].

Um terceiro modelo considerado por vários algoritmos de transmissão de ordem total é o modelo *assíncrono temporizado* definido por Cristian e Fetzer [1999]. Em sua forma mais simples, esse modelo pode ser visto como um modelo assíncrono com a noção de tempo físico e uma suposição de que "a maioria das mensagens provavelmente chegará ao seu destino dentro de um atraso conhecido ÿ" [Cristian et al. 1997; Cristian e Fetzer 1999].

5.2. Resultados de Impossibilidade

Há um resultado teórico importante relacionado ao problema do consenso (ver nota de rodapé 10). Foi provado que não há solução determinística para o problema de consenso em sistemas assíncronos se apenas um único processo pode travar [Fischer et al. 1985]. Dolev et ai. [1987] mostraram que a transmissão de ordem total pode ser transformada em consenso, provando assim que a impossibilidade de consenso também vale para a transmissão de ordem total. Esses resultados de impossibilidade foram a motivação para estender o sistema assíncrono com a introdução de *oráculos* para tornar o consenso e a transmissão de ordem total solucionáveis.11

5.3. Oráculos

Em suma, um oráculo (distribuído) pode ser visto como algum componente que os processos podem consultar. Um oráculo fornece informações que os algoritmos podem usar para guiar suas escolhas. Os oráculos mais frequentemente considerados em sistemas distribuídos são detectores de falhas e cara ou coroa. Uma vez que as informações fornecidas por esses oráculos tornam o consenso e a transmissão de ordem total solucionáveis, eles aumentam o poder do modelo de sistema assíncrono.

5.3.1. Detectores de falha. Um detector de falhas é um oráculo que fornece informações sobre o status atual dos processos, por exemplo, se um determinado processo travou ou não.

A noção de detectores de falha foi formalizada por Chandra e Toueg [1996]. Resumidamente, um detector de falhas é modelado como um conjunto de módulos distribuídos, um módulo *FDi* anexado a cada processo *pi.* Qualquer processo *pi* pode consultar seu módulo detector de falhas *FDi* sobre o status de outros processos.

Detectores de falhas podem não ser *confiáveis,* no sentido de que fornecem informações que nem sempre correspondem ao estado real do sistema. Por exemplo, um módulo detector de falha *FDi* pode fornecer a informação errônea de que algum processo *pj* travou enquanto, na realidade, *pj* está correto e em execução. Por outro lado, *FDi* pode fornecer a informação de que um processo *pk* está correto, enquanto *pk* realmente travou.

Para refletir a falta de confiabilidade das informações fornecidas pelos detectores de falha, dizemos que um processo *pi suspeita* de algum processo *pj* sempre que *FDi,* o módulo detector de falhas anexado a *pi,* retorna a informação (não confiável) de que *pj* travou. Em outras palavras, uma suspeita é uma crença (por exemplo, *"pi* acredita que *pj* caiu") em oposição a um fato conhecido (por exemplo, *"pj* caiu e *pi* sabe disso").

Existem várias classes de detectores de falhas, dependendo de quão pouco confiáveis podem ser as informações fornecidas pelo detector de falhas. As classes são definidas por duas propriedades, chamadas de *integridade* e *precisão,* que restringem a gama de possíveis erros. Neste artigo, consideramos quatro classes diferentes de detectores de falha, chamados P (perfeito), P (eventualmente perfeito), S (forte) e S (eventualmente forte).
As quatro classes compartilham a mesma propriedade de completude e diferem apenas por sua propriedade de precisão [Chandra e Toueg 1996]:

(FORTE INTEGRALIDADE) Eventualmente, todo processo defeituoso é permanentemente suspeito por todos os processos corretos.

(FORTE PRECISÃO) Nenhum processo é suspeito antes de travar. [classe P]

(EVENTUAL FORTE PRECISÃO) Há um tempo após o qual processos corretos não são suspeitos por nenhum processo correto. [classe P]

11Chandra e Toueg [1996] mostram que o consenso pode ser transformado em transmissão de ordem total. O resultado vale também para falhas arbitrárias. Assim, consenso e broadcast de ordem total são problemas equivalentes, ou seja, se existe um algoritmo que resolve um problema, então ele pode ser transformado em um algoritmo que resolve o outro problema.

(PRECISÃO FRACA) Alguns processos nunca são suspeitos. [classe S]

(EVENTUAL PRECISÃO FRACA) Há um tempo após o qual algum processo correto nunca é suspeito por nenhum processo correto. [classe S]

Um detector de falhas de classe S com uma maioria de processos corretos nos permite resolver o consenso [Chandra e Toueg 1996]. Além disso, Chandra et al. [1996] mostraram que um detector de falha da classe S é o detector de falha mais fraco que nos permite resolver o consenso.12

5.3.2. Oráculo aleatório. Outra abordagem para estender o poder do modelo de sistema assíncrono é introduzir a capacidade de gerar valores aleatórios.

Por exemplo, os processos podem ter acesso a um módulo que gera um bit aleatório quando consultado (ou seja, uma variável aleatória de Bernoulli).

Essa abordagem é usada por uma classe de algoritmos chamados algoritmos aleatórios. Esses algoritmos podem resolver problemas como consenso (e, portanto, transmissão de ordem total) de maneira probabilística. A probabilidade de que tais algoritmos terminem antes de algum tempo t, vai para um, como *t* vai para o infinito (por exemplo, Ben-Or [1983] e Chor e Dwork [1989]). Observe que resolver um problema de forma determinística e resolvê-lo com probabilidade 1 não são a mesma coisa.

5.4. Uniformidade de graça

Na Seção 2, explicamos a diferença entre especificações *uniformes* e *não uniformes* . Guerraoui [1995] mostra que qualquer algoritmo que resolva Consenso com P (S, S, respectivamente), também resolve Consenso Uniforme com P (S, S, respectivamente).

É fácil mostrar que esse resultado também vale para a transmissão de ordem total. Suponha que exista um algoritmo que resolva a transmissão de ordem total não uniforme (Acordo não uniforme, Ordem total não uniforme) com P, S ou S, mas não resolve a transmissão de ordem total uniforme. Usando a transformação da ordem total transmitida em consenso (ver Seção 5.2), esse algoritmo pode ser usado para obter um algoritmo que resolva o consenso não uniforme, mas não o consenso. Isso está em contradição com Guerraoui [1995]. Portanto, impor uniformidade não tem custo adicional nos modelos assíncronos com detectores de falha P, S e S.

Observe, no entanto, que o resultado não é válido para algoritmos de transmissão de ordem total que dependem de um detector de falha perfeito (P) ou quase perfeito (consulte a Seção 5.5).

5.5. Falha controlada por processo

A falha controlada pelo processo é a capacidade dada aos processos de matar outros processos ou cometer suicídio. Em outras palavras, essa é a capacidade de forçar artificialmente a falha de um processo. Permitir a falha controlada pelo processo em um modelo de sistema aumenta seu poder. Na verdade, isso permite transformar falhas graves (por exemplo, omissão, bizantina) em falhas menos graves (por exemplo, colisão) e emular um detector de falhas “quase perfeito”. No entanto, esse poder não vem sem um preço.

Transformação automática de falhas. Neiger e Toueg [1990] apresentam uma técnica que usa travamento controlado por processo para transformar falhas graves (por exemplo, omissão, bizantina) em menos graves (por exemplo, falhas de travamento). Em suma, a técnica se baseia na ideia de que os processos têm seu comportamento monitorado. Então, sempre que um processo começa a se comportar incorretamente (por exemplo, omissão, bizantino), ele é encerrado.13

No entanto, esta técnica não pode ser utilizada em sistemas com canais com perdas ou sujeitos a partições. De fato, em tais contextos, os processos podem acabar matando uns aos outros até que nenhum deles seja deixado vivo no sistema.

Emulação de um detector de falhas quase perfeito. Um detector de falha perfeito (P) satisfaz tanto integridade forte quanto precisão forte (nenhum processo é suspeito antes

12 O detector de falha mais fraco para resolver o consenso é geralmente dito ser W, que difere de S por satisfazer uma propriedade de completude fraca em vez de completude forte. No entanto, Chandra e Toueg [1996] provam a equivalência de S e W.

13A técnica real é mais complexa do que a descrita aqui, mas isso dá a ideia básica.

trava [Chandra e Toueg 1996]). Em sistemas práticos, detectores de falhas perfeitos são extremamente difíceis de implementar devido à dificuldade em distinguir processos travados de processos muito lentos. A ideia da emulação é simples: sempre que um detector de falha suspeitar de um processo p, então *p* é morto (forçado a travar). Fetzer [2003] propõe uma emulação diferente, baseada em watchdogs confiáveis, para garantir que nenhum processo seja suspeito antes de travar.

Custo de um almoço grátis. A falha controlada pelo processo tem um preço. Um algoritmo tolerante a falhas só pode tolerar a falha de um número limitado de processos. Em um sistema com travamento controlado por processo, esse limite inclui não apenas falhas genuínas, mas também falhas provocadas por travamento controlado por processo. Isso significa que cada falha provocada *diminui* efetivamente o número de falhas genuínas que podem ser toleradas, degradando assim a tolerância real a falhas do sistema.

6. MECANISMOS DE TOLERÂNCIA A FALHAS

Os algoritmos de broadcast de ordem total descritos na Seção 4 não são tolerantes a falhas: se um único processo falhar, as propriedades especificadas na Seção 2.3 não serão satisfeitas. Para serem tolerantes a falhas, os algoritmos de broadcast total ou der dependem de várias técnicas apresentadas nesta seção.
Observe que é difícil discutir essas técnicas sem entrar em detalhes de implementação específicos. No entanto, tentamos manter a discussão o mais geral possível. Observe também que os algoritmos podem realmente combinar várias dessas técnicas, por exemplo, detecção de falhas (Seção 6.1) com padrões de comunicação resilientes (Seção 6.3).

6.1. Detecção de Falha

Um padrão recorrente em todos os algoritmos distribuídos é que um processo *p* espera por uma mensagem de algum outro processo q. Se *q* falhar, o processo *p* é bloqueado. A detecção de falhas é um mecanismo básico para evitar que *p* seja bloqueado.

A detecção de falha não confiável foi formalizada por Chandra e Toueg [1996] em termos de duas propriedades: *precisão* e *integridade* (ver Seção 5.3.1). A completude evita o problema de bloqueio que acabamos de mencionar. A precisão evita que os algoritmos funcionem para sempre sem resolver o problema.

Detectores de falha não confiáveis podem ser muito fracos para alguns algoritmos de transmissão de ordem total que requerem informações *confiáveis* de detecção de falha fornecidas por um detector de falha *perfeito ,* conhecido como P (consulte a Seção 5.5).

6.2. Serviço de associação de grupo

O mecanismo de detecção de falha de baixo nível não é a única maneira de abordar o problema de bloqueio mencionado na seção anterior. O bloqueio também pode ser evitado contando com um mecanismo de nível superior, ou seja, um *serviço de associação de grupo.*

Um serviço de associação de grupo é um serviço distribuído responsável por gerenciar a associação de grupos de processos (consulte a Seção 3.4 e a pesquisa de Chockler, Keidar e Vitenberg [2001]).
As participações sucessivas de um grupo são chamadas de *visualizações* do grupo. Sempre que a associação é alterada, os relatórios de serviço são alterados para todos os membros do grupo, fornecendo a eles a nova exibição.

Um serviço de associação de grupo geralmente fornece integridade forte: se um processo *p* membro de algum grupo *G* falhar, o serviço de associação fornece aos membros sobreviventes de *G* uma nova visão da qual *p* é excluído. No modelo de partição primária (consulte a Seção 3.4), a precisão das notificações de falha é garantida forçando o travamento de processos que foram incorretamente suspeitos e excluídos da associação, um mecanismo denominado travamento controlado por processo (consulte a Seção 5.5) . .

Além disso, no modelo de partição primária, o serviço de associação de grupo fornece notificações consistentes aos membros do grupo: as visualizações sucessivas de um grupo são notificadas na mesma *ordem* para todos os seus membros.

Para resumir, enquanto os detectores de falha fornecem notificações de falha inconsistentes, um serviço de associação de grupo fornece

notificações de falha consistentes. Além disso, algoritmos de ordem total que dependem de um serviço de associação de grupo para tolerância a falhas, exploram outra propriedade que geralmente é fornecida junto com o serviço de associação de membros, ou seja, *sincronia de visualização* (consulte a Seção 3.3). Grosso modo, a sincronia de visualização garante que, entre duas visualizações sucessivas *v* e *v ,* os processos na interseção *v* ÿ *v* entregam o mesmo conjunto de mensagens. O serviço de associação de grupo e a sincronia de exibição foram usados para implementar sistemas complexos de comunicação de grupo (por exemplo, Isis [Birman e van Renesse 1994], Totem [Moser et al. 1996], Transis [Dolev e Malkhi 1994, 1996; Amir et al. 1992], Phoenix [Malloth et al.

1995; Malloth 1996]).

6.3. Padrões de comunicação resilientes

Conforme mostrado nas seções anteriores, um algoritmo pode contar com um mecanismo de detecção de falhas ou com um serviço de associação de grupo para evitar o problema de bloqueio. Para ser tolerante a falhas, outra solução é evitar qualquer padrão de bloqueio potencial.

Considere, por exemplo, um processo *p* esperando por nÿ *f* mensagens, onde *n* é o número de processos no sistema e *f* o número máximo de processos que podem travar. Se todos os processos corretos enviarem uma mensagem para p, então o padrão acima é não bloqueante. Chamamos esse padrão de padrão *resiliente* . Se um algoritmo usa apenas padrões resilientes, ele evita o problema de bloqueio *sem* usar nenhum mecanismo de detecção de falha ou serviço de associação de grupo. Tais algoritmos foram, por exemplo, propostos por Rabin [1983], Ben-Or [1983] e Pedone et al. [2002] (os dois primeiros são algoritmos de consenso, ver nota de rodapé 10).

6.4. Estabilidade de mensagem

Evitar o bloqueio não é o único problema que os algoritmos de transmissão de ordem total tolerantes a falhas precisam resolver. A Figura 1 ilustra uma violação da propriedade do Acordo Uniforme. Observe que esse problema não está relacionado ao bloqueio.

O mecanismo que resolve o problema é chamado de *estabilidade da mensagem.* a mes

o sábio *m* é considerado *k-estável,* se *m* foi recebido por *k* processos. Em um sistema no qual no máximo *f* processos podem travar, *f* +1-estabilidade é a propriedade importante a ser detectada. Se alguma mensagem *m* é *f* +1-estável, então *m* é recebida por pelo menos um processo correto. Com tal garantia, um algoritmo pode garantir facilmente que *m* seja recebido por todos os processos corretos. *f* +1-estabilidade é muitas vezes chamada simplesmente *de estabilidade.* A detecção de estabilidade é geralmente baseada em algum esquema de reconhecimento ou passagem de token.

Outro uso para a estabilidade da mensagem é a recuperação de recursos. De fato, quando um processo detecta que uma mensagem se tornou estável em todo o sistema, ele pode liberar recursos associados a ela.

mensagem.

6.5. Consenso

Os mecanismos descritos até agora são mecanismos de baixo nível nos quais os algoritmos de broadcast totais tolerantes a falhas podem se basear.

Outra opção para um algoritmo de transmissão de ordem total tolerante a falhas é contar com mecanismos de alto nível que resolvam todos os problemas relacionados à tolerância a falhas (ou seja, os problemas mencionados anteriormente).
O problema do consenso (ver nota de rodapé 10) é um desses mecanismos. Alguns algoritmos resolvem a transmissão de ordem total reduzindo-a a um problema de consenso. Dessa forma, a tolerância a falhas, incluindo detecção de falhas e detecção de estabilidade de mensagens, fica oculta na abstração de consenso.

6.6. Mecanismos para canais com perdas

Além dos mecanismos usados para tolerar falhas de processo, precisamos dizer algumas palavras sobre os mecanismos para tolerar falhas de canal. Primeiro, deve-se mencionar que vários algoritmos de transmissão de ordem total evitam o problema confiando em alguma camada de comunicação que cuide da perda de mensagem (ou seja, esses algoritmos como canais confiáveis e, portanto, não discutem a perda de mensagem). Em contraste, outros algoritmos são construídos diretamente sobre os canais com perdas e, portanto, lidam com a perda de mensagens explicitamente.

Para lidar com a perda de mensagens, a solução padrão é contar com um mecanismo de confirmação positivo ou negativo.
Com confirmação positiva, o recebimento de mensagens é confirmado; com confirmação negativa, a detecção de uma mensagem ausente é sinalizada. Os dois esquemas podem ser combinados.

Algoritmos baseados em token (isto é, sequenciador de movimento ou algoritmos baseados em privilégios) dependem da passagem de token para detectar perdas de mensagens: o token pode ser usado para transmitir confirmações ou para detectar mensagens ausentes. Portanto, os algoritmos baseados em token usam o token para fins de pedido, mas também para implementar canais confiáveis.

Tabela II. Abreviaturas Usadas nas Tabelas III–V

	sim	
	um pouco	explicado no texto
×	*não*	
especificação	*especial*	explicado no texto
inf.	*informal*	explicado no texto
NS	*não especificado*	também significa "não discutido"
n/a	*não aplicável*	
+	uma *confirmação positiva*	
ÿ	uma *confirmação negativa*	
Membro do grupo	GM	
Detector/ detecção de falha	FD	
Contras.	*padrões de comunicação resilientes de consenso*	
RCP	ByzA. *acordo bizantino*	

7. LEVANTAMENTO DOS ALGORITMOS EXISTENTES

Esta seção fornece uma extensa pesquisa de algoritmos de transmissão de ordem total. Apresentamos cerca de sessenta algoritmos publicados em revistas científicas ou anais de conferências nas últimas três décadas. Fizemos todos os esforços possíveis para sermos exaustivos e estamos bastante confiantes de que este artigo apresenta uma boa imagem do campo no momento em que foi escrito. No entanto, devido ao fluxo contínuo de artigos sobre o assunto, podemos ter esquecido um ou dois algoritmos.

Nas Tabelas III–V, apresentamos uma visão geral condensada de todos os algoritmos pesquisados, na qual resumimos as características importantes de cada algoritmo. As Tabelas apresentam apenas informações factuais sobre os algoritmos conforme aparecem nos artigos relevantes. Em particular, as Tabelas não apresentam informações resultantes de extrapolação ou dedução não óbvia; a exceção é quando tivemos que interpretar informações para superar diferenças de terminologia. Além disso, as propriedades discutidas no artigo original, mas que não se mostraram corretas, são relatadas como "informais" nas Tabelas. Por uma questão de concisão, vários símbolos e abreviações foram usados ao longo das Tabelas. Eles são explicados na Tabela II. Para cada algoritmo, as Tabelas III–V fornecem as seguintes informações:

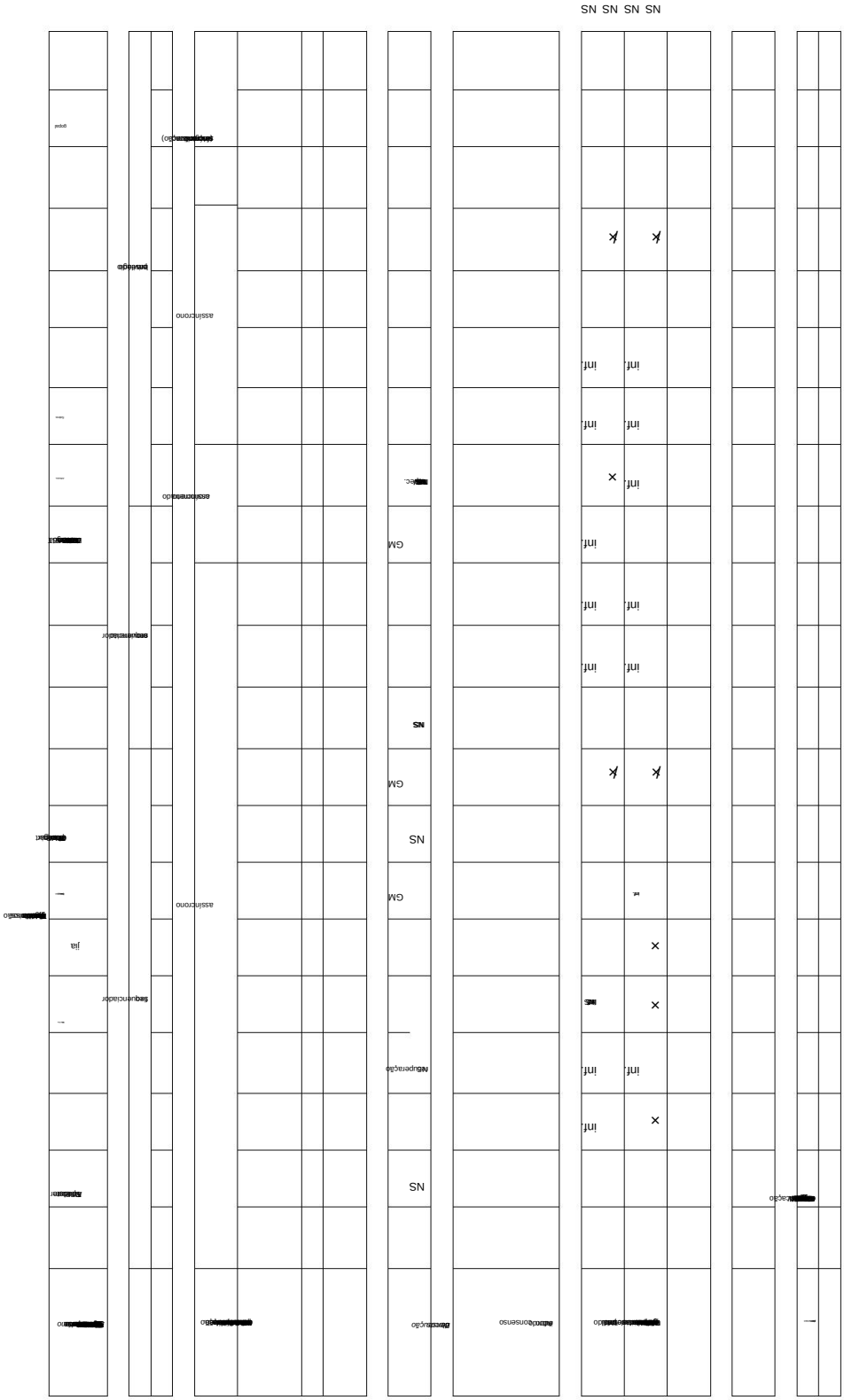
(1) *Informações gerais,* ou seja, o mecanismo de ordenação (ver Seção 4), e se o mecanismo é baseado no tempo ou não (Seção 4.6).

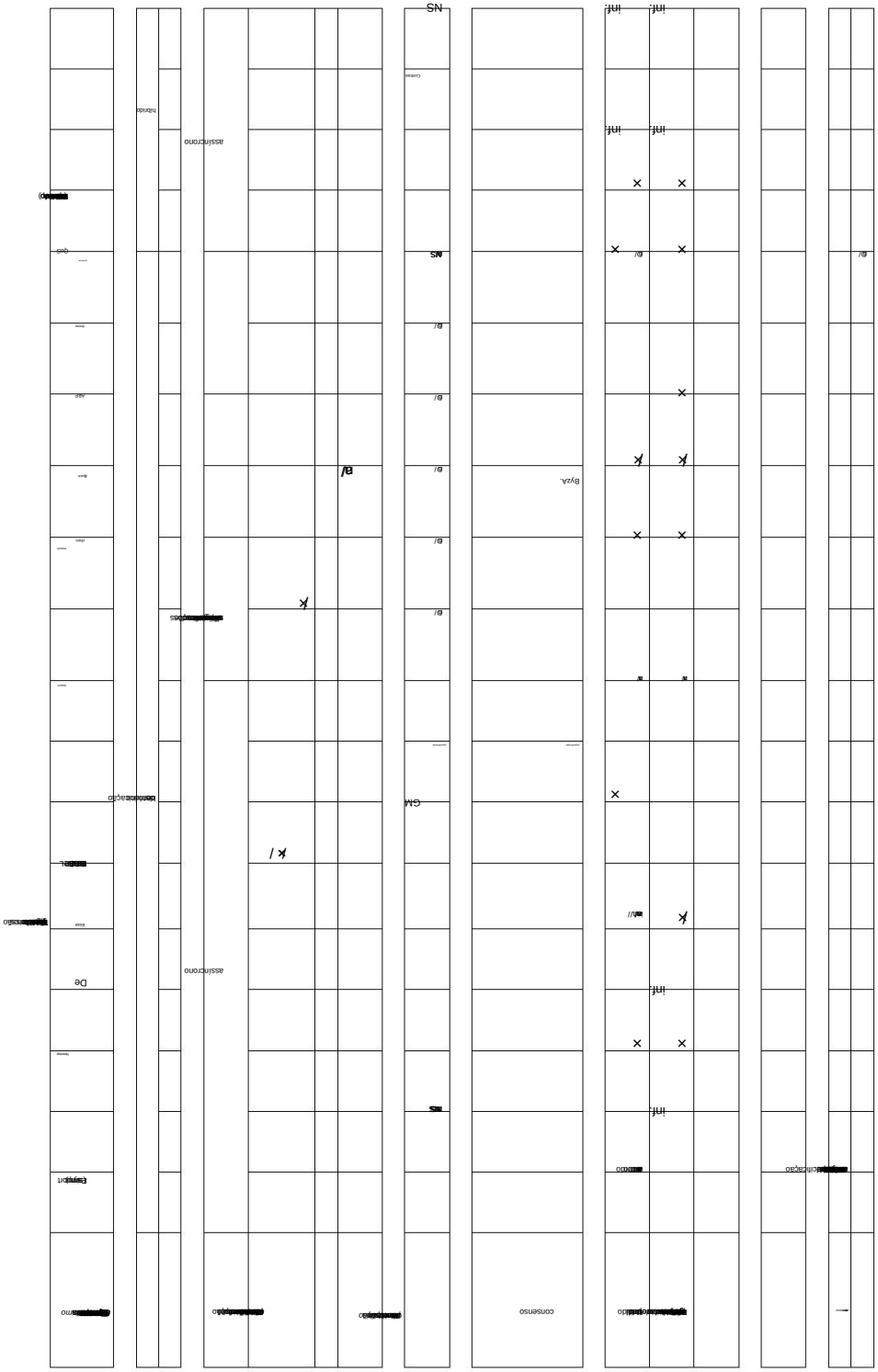
(2) As linhas *de informações gerais* são seguidas por linhas que descrevem as suposições sobre as quais o algoritmo se baseia, ou seja, o que é *fornecido* a ele: (a) As linhas do

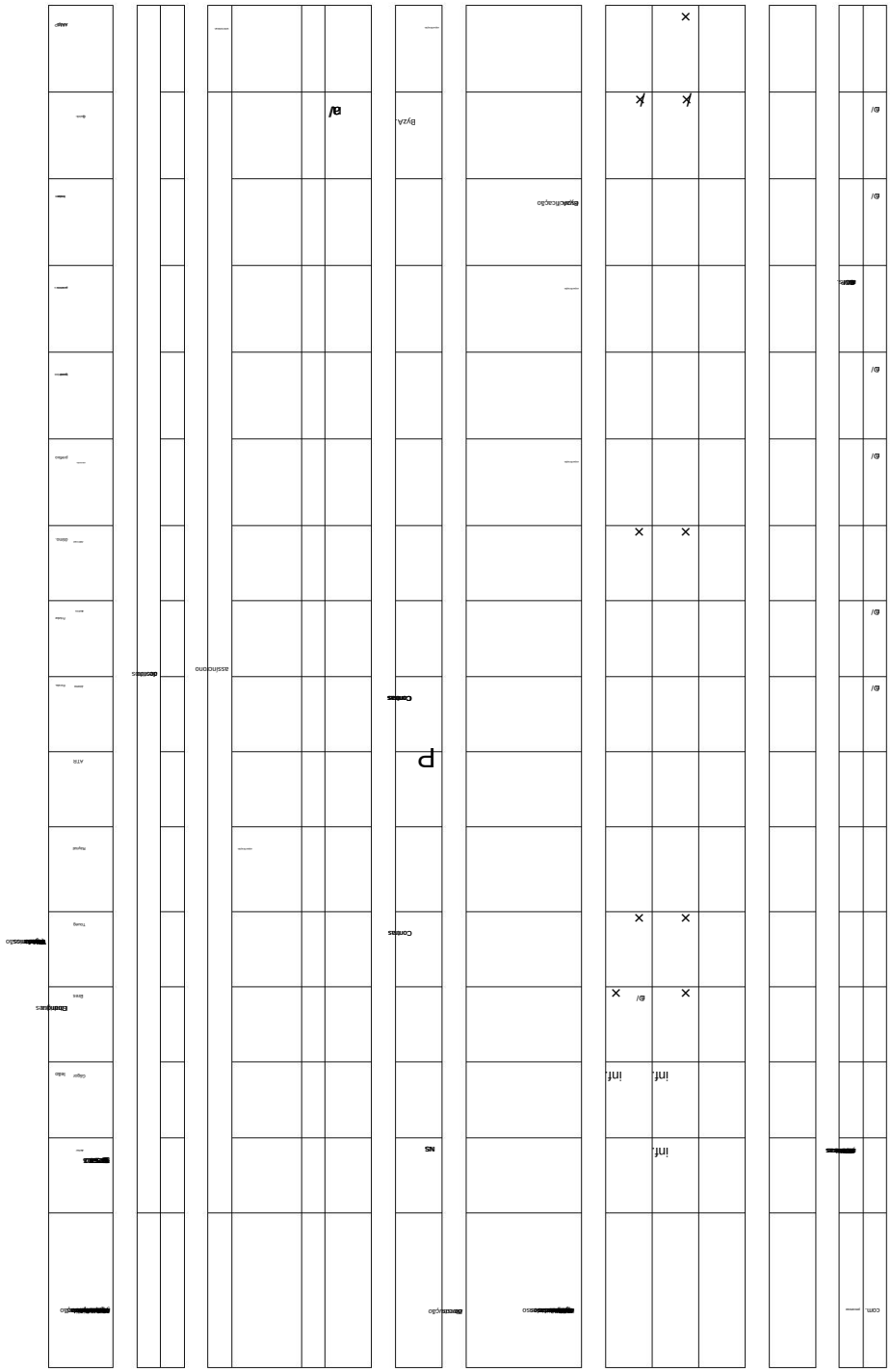
modelo do sistema especificam as suposições de sincronia, as suposições feitas sobre falhas de processo (linhas: *falha, omissão, bizantino)* e canais de comunicação (linhas: *confiável, FIFO).* Canais confiáveis garantem que se um processo correto *p* enviar uma mensagem *m* para um processo correto q, então *q* receberá *m* [Aguilera et al. 1999]. A linha *particionável* indica se o algoritmo funciona ou não com semântica de associação particionável (consulte a Seção 3.4). Em particular, os algoritmos nos quais apenas os processos em uma partição primária podem funcionar não são considerados particionáveis. (b) As linhas denominadas *Condição para vivacidade* discutem as suposições necessárias para

garantir a vivacidade do algoritmo: —A linha *live... X* significa que a vivacidade do algoritmo requer a vivacidade do bloco de construção X (no qual o algoritmo depende).

Por exemplo, *ao vivo... GM* significa que o algoritmo está ativo, se o bloco de construção de associação de grupo no qual o algoritmo se baseia estiver ativo.







—A linha *other* acrescenta o seguinte na formação: *NS = não especificado* significa que vivacidade não é discutida no artigo; *n/a = não aplicável* significa que nenhuma suposição adicional é necessária para garantir a vivacidade (isso se aplica principalmente a algoritmos que assumem um modelo síncrono); = *um pouco* e *especificação. = especial* refere-se a uma discussão sobre vivacidade mais adiante no artigo; *recovery* significa que o algoritmo está bloqueando, ou seja, liveness requer a recuperação de processos travados; P/S refere-se ao detector de falha necessário para garantir a vivacidade.

(c) O próximo grupo de linhas indica o *(s) bloco(s) de construção* usado(s) pelo algoritmo. Os blocos de construção considerados são: *sincronia de visão* (Seção 6.2), que engloba um serviço *de associação de grupo ; transmissão confiável* (Seção 2.3) *transmissão causal* (Seção 2.6.2); *consenso* (Seção 4.5); ou *outro. Outro* pode ser ByzA. = *Acordo bizantino*[14] ou *spec.=especial,* o que significa que a explicação está no texto.

(3) Depois de discutir o que é fornecido *"para"* os algoritmos, discutimos o que é fornecido *"pelos"* algoritmos.

(a) As primeiras linhas fornecem as *Propriedades garantidas* pelos algoritmos. Conforme discutido na Seção 2, a transmissão de ordem total é especificada pelas seguintes propriedades: Validade, Acordo Uniforme, Integridade Uniforme, Ordem Total Uniforme. Validade e Integridade Uniforme não aparecem nas Tabelas. A razão é que essas propriedades raramente são discutidas nos artigos (os autores geralmente assumem que são trivialmente asseguradas).

Primeiro discutimos Acordo e Acordo Uniforme, depois Pedido Total e Pedido Total Uniforme. Por fim, mencionamos se o algoritmo garante adicionalmente a ordem FIFO ou a ordem causal. Em todas essas entradas, seria de esperar um *sim* ou um *não.*
Infelizmente, muitos artigos não fornecem provas (geralmente apenas argumentos informais), o que significa que essas propriedades podem ser questionadas. Neste caso, *inf. = informal* aparece na Tabela. Se um algoritmo não discute as propriedades da difusão total da ordem, a entrada correspondente menciona *NS = não especificado.* Se a propriedade não uniforme for discutida apenas informalmente, então a entrada correspondente para a propriedade uniforme é deixada em branco (em uma discussão informal, a distinção entre propriedade uniforme e não uniforme geralmente não aparece). */× (=sim/não)* aparece em algumas entradas para a propriedade uniforme, significando que esses algoritmos fornecem vários níveis de Qualidade de Serviço (QoS), que incluem uma versão uniforme e uma versão não uniforme do algoritmo, onde a versão não uniforme é mais eficiente. Além disso, para poder comparar algoritmos não particionáveis com algoritmos particionáveis, consideramos as propriedades impostas pelos primeiros, quando executados em um modelo de sistema não particionável.

Para as linhas *FIFO order* e *causal order, = yes* aparece apenas se esta característica estiver explícita no papel. Caso contrário, a entrada é simplesmente deixada em branco. Finalmente, se um algoritmo não é tolerante a falhas, então a distinção entre as propriedades uniformes e não uniformes não faz sentido. Neste caso, a entrada menciona *n/a = não aplicável.*

(b) As linhas chamadas *de grupos de destino* indicam se o algoritmo suporta a transmissão de ordem total de uma mensagem para vários grupos (múltiplos *de linha)* e se os algoritmos suportam a transmissão *aberta*

14No problema do acordo bizantino, também comumente conhecido como o "problema dos generais bizantinos" [Lamport et ai. 1982], todo processo tem um conhecimento *a priori* de que um determinado processo *s* deve transmitir uma única mensagem m. Informalmente, o problema requer que todos os processos corretos entreguem a mesma mensagem, que deve ser m, se o remetente *s* estiver correto. Como o nome indica, o acordo bizantino foi estudado principalmente em relação às falhas bizantinas.

grupos (consulte a Seção 3). A entrada é deixada em branco se a questão não for discutida explicitamente no documento.

(4) O último grupo de linhas, chamado de *mecanismos tolerantes a falhas,* discute os mecanismos usados para fornecer tolerância a falhas. O *processo de* linha menciona os mecanismos usados para tolerar falhas de processo (consulte a Seção 6). Observe que alguns desses mecanismos tolerantes a falhas também aparecem como *blocos de construção.* No entanto, nem todos os blocos de construção foram relatados como mecanismos tolerantes a falhas (por exemplo, broadcast confiável, broadcast causal).15

A linha *comm.* menciona os mecanismos usados para lidar com perdas de mensagens.
A maioria dos algoritmos assume canais confiáveis subjacentes, caso em que a entrada menciona *n/ a = não aplicável.* As siglas *+a* e *ÿa* indicam um mecanismo de reconhecimento positivo, respectivamente negativo. As outras entradas são *flood* (flooding), *special* (explicação no texto abaixo) e GM = *group member ship.* No contexto de canais não confiáveis, o mecanismo GM é usado no caso de algum processo *p* esperar por uma mensagem de algum outro processo *q* e se nenhuma mensagem for recebida (por exemplo, devido a perda), então *p* solicita a exclusão de *q* dos membros.

Nas Seções 7.1 a 7.6, damos uma breve descrição de cada algoritmo individual para complementar as informações fornecidas nas Tabelas. Ao contrário das Tabelas, as descrições de texto também apresentam informações que deduzimos dos documentos relevantes. Em alguns casos, a falta de detalhes técnicos sobre os algoritmos (em particular, no caso de falhas) nos leva a *extrapolar* seu comportamento. Neste caso, tentamos evitar ser muito assertivos (por exemplo, usando o condicional) e gentilmente recomendamos que o leitor trate esta informação especulativa com um grau apropriado de ceticismo.

15A decisão sobre o que é um mecanismo de tolerância a falhas e o que não é é um tanto arbitrária. Decidimos manter baixo o número de mecanismos mencionados na Seção 6, ou seja, mencionar apenas os mecanismos-chave.

Pensamos que é útil enfatizar novamente as respectivas funções das Tabelas e o texto que as acompanha nas Seções 7.1 a 7.6. As Tabelas fornecem informações *factuais* sobre cada algoritmo, conforme publicado nos artigos relevantes. Em contraste, o texto fornece informações complementares, incluindo informações que extrapolamos . Em particular, o texto explica a originalidade de cada algoritmo e complementa os itens deixados vagos nas Tabelas (ou seja, esses pontos são vagos no próprio artigo). Especificamente, para alguns dos algoritmos, as propriedades relatadas nas Tabelas são mais fracas do que aquelas que o algoritmo pode garantir. Nesse caso, o texto menciona (e discute) a propriedade mais forte que pode existir. Enfatizamos este ponto, pois a má compreensão das respectivas funções do texto e das Tabelas pode levar à impressão errônea de que o texto e as Tabelas estão em contradição.

7.1. Algoritmos de sequenciador fixo

Independentemente da variante que eles adotam (ver Seção 4.1), todos os algoritmos sequenciadores assumem um modelo de sistema assíncrono e usam ordenação livre de tempo. Eles toleram falhas de colisão, exceto Rampart, que também tolera falhas bizantinas.
Todos eles contam com travamento controlado por processo para lidar com falhas; explicitamente (por exemplo, Tandem) ou por meio de associação e exclusão de grupo (por exemplo, Isis, Rampart).

7.1.1. Ameba. O sistema de comunicação de grupo Amoeba [Kaashoek e Tanenbaum 1996] suporta algoritmos das duas primeiras variantes de algoritmos de sequenciador fixo. A primeira corresponde à variante UB (unicast-broadcast) ilustrada na Figura 6(a) (Seção 4.1). A segunda variante corresponde a BB (broadcast broadcast), ver Figura 6(b). As duas variantes compartilham as mesmas propriedades.

O Amoeba assume canais com perdas e implementa a retransmissão de mensagens como parte do algoritmo de transmissão de ordem total. A Amoeba usa uma combinação de reconhecimentos positivos e negativos. O

protocolo atual é bastante complexo porque é combinado com controle de fluxo e também tenta minimizar o custo de comunicação.

Ameba tolera falhas usando um serviço de associação de grupo. Os processos suspeitos são excluídos do grupo em decorrência da decisão unilateral de um único processo.

As propriedades dos algoritmos Amoeba são apenas discutidas informalmente no artigo. No entanto, como as mensagens são entregues antes de serem estáveis, o algoritmo só pode satisfazer as propriedades não uniformes de Acordo e Pedido Total.

7.1.2. MTP. MTP [Armstrong et al. 1992] é um algoritmo projetado principalmente para streaming de vídeo e aplicativos multimídia semelhantes. O algoritmo assume que o sistema não é uniforme com relação à probabilidade de falhas do processo. Em particular, assume que um processo, denominado processo mestre, nunca falha. O mestre é então designado como sequenciador, e os protocolos seguem a variante UUB (unicast-unicast-broadcast, ver Figura (c) 6). Quando um processo *p* tem uma mensagem *m* para transmitir, *p* solicita um número de seqüência para *m* do seqüenciador. Uma vez obtido o número de seqüência, ele envia m, juntamente com o número de seqüência, para todos os destinos e para o mestre. Ao mesmo tempo, os processos de destino aprendem sobre o status das mensagens anteriores e entregam aquelas que foram aceitas pelo mestre.

O protocolo tolera falhas de travamento dos processos de destino e remetentes, pois todas as partes que envolvem as decisões são executadas pelo mestre. A falha do mestre é brevemente discutida no final do artigo. Os autores sugerem que o mestre pode se tornar mais resiliente introduzindo redundância e usando técnicas de replicação.

7.1.3. Tandem. O protocolo de atualização global Tandem [Carr 1985] é um algoritmo sequenciador fixo da variante UUB (ver Figura 6(c)). O algoritmo permite que, no máximo, uma mensagem de aplicação seja ampla lançado de cada vez e, portanto, não precisa de números de sequência. Mais tarde, Cristian et al. [1994] descrevem uma variante UB de Tandem que permite transmissões simultâneas (e, portanto, precisa de números de sequência).

7.1.4. Garcia-Molina e Spauster. O algoritmo proposto por Garcia-Molina e Spauster [1991] é baseado em um grafo de propagação (uma floresta) para suportar múltiplos grupos sobrepostos. O grafo de propagação é construído de tal forma que a cada grupo é atribuído um nó inicial. Os remetentes enviam suas mensagens para os nós iniciais correspondentes e as mensagens viajam ao longo das bordas do grafo de propagação. Ou decisões importantes são resolvidas ao longo do caminho. Quando usado em uma única configuração de grupo, o algoritmo se comporta como outros algoritmos de sequenciador fixo (ou seja, o grafo de propagação é uma árvore de profundidade 1).

O algoritmo assume um modelo assíncrono e requer relógios sincronizados. No entanto, os relógios sincronizados são necessários apenas para limitar o comportamento do algoritmo quando ocorrem falhas de travamento. Nem o mecanismo de ordenação nem o mecanismo de tolerância a falhas realmente precisam deles.

Em caso de falhas, o algoritmo se comporta de maneira não convencional. De fato, se um processo não-folha *p* falhar, seus descendentes no grafo de propagação não receberão nenhuma mensagem até que *p* tenha se recuperado. Portanto, o algoritmo tolera falhas de processo apenas se for garantido que esses processos serão eventualmente recuperados.

7.1.5. Jia. Jia [1995] propôs outro algoritmo baseado em grafos de propagação, que cria grafos mais simples do que o algoritmo de Garcia-Molina e Spauster [1991] (ver Seção 7.1.4). Infelizmente, o algoritmo originalmente proposto por Jia [1995] está incorreto. Chiu e Hsiao [1998] fornecem uma correção para o algoritmo que funciona apenas em um modelo mais restrito (ou seja, apenas para grupos fechados). Além disso, Shieh e Ho [1997] fornecem uma correção para a complexidade da mensagem calculada por Jia [1995].

O algoritmo de Jia baseia-se na noção de metagrupos, definida no artigo como “o conjunto de processos que têm exatamente os mesmos grupos de membros” (ou seja, o conjunto de processos que pertencem exatamente ao mesmo conjunto de grupos de destino). Os metagrupos são organizados em árvores de propagação, de acordo com os membros que representam. O fluxo de mensagens é feito ao longo das árvores, criando assim a ordem de entrega.

Jia [1995] descreve uma forma de mecanismo de associação de grupo que é usado para redefinir as partes do grafo de propagação que devem mudar quando um processo é excluído. Jia também sugere que, ao contrário do algoritmo de Garcia-Molina e Spauster [1991] (Seção 7.1.4), os nós na árvore consistem em metagrupos inteiros, em vez de processos únicos. Assim, as mensagens não seriam interrompidas, a menos que todos os membros de um metagrupo intermediário falhassem. A questão é, no entanto, abordada apenas informalmente.

7.1.6. Ísis (Sequenciador). Birman et ai. [1991] descreve várias primitivas de transmissão do sistema Isis, incluindo uma primitiva de transmissão de ordem total chamada ABCAST.
A primitiva ABCAST é implementada usando um algoritmo sequenciador fixo (diferente do algoritmo usado em versões anteriores do sistema; veja a Seção 7.5.1).
O algoritmo Isis (sequenciador) é um algoritmo sequenciador fixo da variante BB (ver Figura 6 (b)), que usa uma primitiva causal de transmissão ampla. O algoritmo assume falhas de travamento.

Construído sobre uma primitiva de transmissão causal, o algoritmo Isis ABCAST preserva a ordem causal. Além disso, embora o algoritmo não suporte a ordem total para vários grupos sobrepostos, a ordem causal é, no entanto, preservada neste contexto. O algoritmo de transmissão ampla de ordem total garante apenas as propriedades não uniformes de Acordo e Ordem Total.

Para tolerância a falhas, o algoritmo de transmissão de ordem total depende de um serviço de associação de grupo e sincronia de exibição (Seção 6.2).

Por fim, os autores também mencionam brevemente que mudar o papel do sequenciador na ausência de falhas pode ser uma forma de evitar um gargalo. No entanto, a ideia não é desenvolvida.

7.1.7. Navaratnam e outros Navaratnam e outros. [1988] propõem um protocolo de sequenciador fixo da variante UB (ver Figura 6(a)).
A tolerância a falhas do algoritmo depende de um serviço de associação de grupo e da capacidade de excluir processos suspeitos erroneamente. Semelhante à Amoeba (Seção 7.1.1), a decisão de excluir um processo suspeito pode ser tomada unilateralmente por um único processo.

As propriedades desse algoritmo são discutidas informalmente e é fácil ver que ele satisfaz as propriedades não uniformes de Concordância e Ordem Total. Os autores também fazem uma breve observação sugerindo que o algoritmo não garante propriedades uniformes, mas a redação é um pouco ambígua e as informações fornecidas no artigo não são suficientes para verificar essa interpretação.

7.1.8. Fénix. Phoenix [Wilhelm e Schiper 1995] consiste em três algoritmos que fornecem diferentes níveis de garantias. O primeiro algoritmo (ordem fraca) garante apenas a Ordem Total e o Acordo. O segundo algoritmo (strong ou der) garante tanto o Uniform Total Order quanto o Uniform Agreement. Em seguida, o terceiro algoritmo (ordem híbrida) combina ambas as garantias por mensagem.

Os três algoritmos são baseados em um serviço de associação de grupo e sincronia de visualização (consulte a Seção 3.3).

7.1.9. Muralha. Ao contrário de outros algoritmos sequenciadores, que assumem apenas falhas de travamento, o algoritmo de Rampart [Reiter 1994, 1996] é projetado para tolerar falhas bizantinas. Isso diferencia um pouco esse algoritmo dos outros algoritmos do sequenciador.

Rampart assume um modelo de sistema assíncrono com canais FIFO confiáveis e uma infraestrutura de chave pública na qual cada processo inicialmente conhece o

chave pública de todos os outros processos. Além disso, os canais de comunicação devem ser autenticados, de modo que a integridade das mensagens entre dois processos honestos (ou seja, não bizantinos) seja sempre garantida.

Ao contrário da maioria dos primeiros trabalhos sobre falhas bizantinas, Rampart trata processos honestos e bizantinos separadamente. Em particular, o documento define a uniformidade como uma propriedade que se aplica apenas a processos honestos (consulte a Nota 1 na Seção 2.4). Com esta definição, a Rampart satisfaz tanto o Acordo Uniforme quanto o Pedido Total Uniforme.

O algoritmo é baseado em um serviço de associação de grupo, que exige que pelo menos um terço de todos os processos na visão atual cheguem a um acordo sobre a exclusão de algum processo do grupo. Essa condição é necessária porque os processos bizantinos poderiam excluir intencionalmente os processos corretos do grupo.

7.2. Algoritmos do sequenciador

móvel Descrevemos aqui quatro algoritmos do sequenciador móvel, todos livres de tempo. Tanto quanto sabemos, não existe um algoritmo sequenciador de movimento baseado no tempo. Na verdade, é questionável se a ordenação baseada no tempo faria sentido para algoritmos dessa classe.

Os quatro algoritmos se comportam de maneira muito semelhante. Na verdade, três deles – Pinwheel (Seção 7.2.4), RMP (Seção 7.2.2) e DTP (Seção 7.2.3) – são baseados no quarto – o algoritmo de Chang e Maxemchuck [1984] (Seção 7.2.1).)—que cada um deles melhora de uma maneira diferente. O Pinwheel é otimizado para um padrão uniforme de chegada de mensagens, o RMP fornece vários níveis de garantias e o DTP fornece uma detecção mais rápida da estabilidade da mensagem. Os quatro algoritmos também lidam com falhas de processo de maneira muito semelhante, usando um algoritmo de reforma (consulte a Seção 7.2.1). Com exceção do DTP (Seção 7.2.4), todos os algoritmos dependem de um anel lógico ao longo do qual o token circula.

Os quatro algoritmos toleram a perda de mensagens confiando em um protocolo de missão de retransmissão de mensagens que combina e reconhecimentos negativos. Mais precisamente, o token carrega confirmações positivas, mas quando um processo detecta que uma mensagem está faltando, ele envia uma confirmação negativa ao site do token. O esquema de reconhecimento negativo é usado para retransmissão de mensagens, enquanto o esquema positivo é usado para detectar a estabilidade da mensagem.

7.2.1. Chang e Maxemchuck. O algoritmo proposto por Chang e Maxemchuk [1984] é baseado na existência de um anel lógico ao longo do qual um token é passado. O processo que contém o token, também conhecido como token site, é responsável por sequenciar as mensagens que recebe.
A passagem do token serve simultaneamente a dois propósitos: (1) a transmissão do papel do sequenciador e (2) a detecção da estabilidade da mensagem. O ponto (2) requer que o anel lógico abranja todos os processos de destino. Este requisito não é, no entanto, necessário para ordenar mensagens (ponto (1)), e, portanto, o algoritmo se qualifica como um algoritmo baseado em sequenciador de acordo com nossa classificação.

Quando uma falha de processo é detectada (talvez erroneamente) ou quando um processo se recupera, o algoritmo passa por uma fase de reforma. A fase de reforma redefine o anel lógico e elege uma nova inicial para titular do ken. O algoritmo de reforma pode ser visto como uma implementação ad hoc de um serviço de associação de grupo.

As propriedades do algoritmo broadcast de ordem total são discutidas apenas informalmente. No entanto, parece plausível que o algoritmo assegure Pedido Total Uniforme e Acordo Uniforme.

7.2.2. RMP. RMP [Whetten et al. 1994] difere dos outros três algoritmos deste grupo por ser projetado para operar com grupos abertos. Além disso, os autores afirmam que “o RMP fornece vários grupos multicast, em oposição a um único grupo de transmissão”. No entanto, de acordo com sua descrição, o suporte a vários grupos multicast é apenas uma característica associada ao serviço de envio de membros do grupo. É, portanto, duvidoso que

"grupos múltiplos" é usado com o significado de que a ordem total é garantida para processos que estão na interseção de dois grupos (consulte a discussão na Seção 3.2).

Dependendo da escolha do usuário, o RMP satisfaz Acordo, Acordo Uniforme ou nenhuma dessas propriedades. No entanto, para garantir as garantias fortes, o RMP deve assumir que a maioria dos processos permanece correta e sempre conectada. Além disso, o RMP não exclui a contaminação do grupo.

7.2.3. DTP. Conforme mencionado, o DTP [Kim e Kim 1997] difere dos outros algoritmos dessa classe por não depender de um anel lógico para a passagem do token. Em vez disso, o DTP segue uma heurística, onde o token é sempre passado para o processo visto como o menos ativo. Isso garante que as mensagens sejam reconhecidas mais rapidamente quando a atividade (ou seja, mensagens de difusão ampla) não é distribuída uniformemente entre os processos.

7.2.4. Catavento. A originalidade da Pin wheel [Cristian et al. 1997] é que o to ken circula entre os processos a uma velocidade proporcional à atividade global dos processos emissores (ou seja, taxa de transmissão).

O Pinwheel assume que a maioria dos processos permanece correta e conectada o tempo todo (grupo majoritário). O algoritmo é baseado no modelo assíncrono temporizado de Cristian e Fetzer [1999]. Embora dependa de relógios físicos para intervalos, o Pinwheel não precisa assumir que esses relógios estão sincronizados. Além disso, o algoritmo é time-free, pois o tempo não é usado para ordenar mensagens.

A Pinwheel pode garantir o Pedido Total Uniforme, dado um suporte adequado de seus membros do grupo (não detalhado no documento). Pinwheel só satisfaz (não uniforme)
Concordância, mas os autores argumentam que o algoritmo poderia ser facilmente modificado para satisfazer a Concordância Uniforme [Cristian et al. 1997]. Fazer isso exigiria apenas que os processos de destino esperassem até que uma mensagem fosse estável antes de entregá-la.
Os autores afirmam que o algoritmo pré serve à ordem causal, mas isso é válido apenas sob certas restrições que tornam o problema trivial de resolver.16

7.3. Algoritmos baseados em

privilégios Assim como os algoritmos de sequenciador móvel, a maioria dos algoritmos baseados em privilégios são baseados em um anel lógico e, para a maioria deles, dependem de algum tipo de associação de grupo ou protocolo de reconfiguração para lidar com falhas de processo.

7.3.1. Sob demanda. O protocolo On-demand [Cristian et al. 1997], ao contrário de outros algoritmos baseados em privilégios, não depende de um anel lógico. Em vez disso, os processos com uma mensagem a ser transmitida devem obter o token emitindo uma solicitação ao detentor do token atual. Como consequência, o protocolo é mais eficiente se os remetentes enviarem rajadas longas de mensagens e essas rajadas raramente se sobrepõem. Além disso, em contraste com os outros algoritmos, *todos os processos* devem estar cientes da identidade do detentor do token. Assim, a passagem do token é feita por meio de um broadcast.

O protocolo On-demand baseia-se no mesmo modelo do protocolo Pinwheel (Seção 7.2.4). Em outras palavras, ele assume um modelo de sistema assíncrono cronometrado e relógios físicos para timeouts.

Um algoritmo similar, chamado Reqtoken, também é descrito por Friedman e van Re nesse [1997].

7.3.2. Trem. O protocolo [Cristian 1991] é inspirado Trem na imagem de um trem que transporta mensagens e circula entre os processos. Mais concretamente, um token (também conhecido como trem) se move ao longo de um anel lógico e carrega as mensagens. Quando um processo obtém o token, ele recebe as novas mensagens transportadas pelo token, as reconhece e anexa suas próprias mensagens ao token. Em seguida, ele passa o token para o próximo processo. O protocolo Train, onde as mensagens são transportadas pelo token, está em

16Em sistemas com um único grupo fechado, onde os processos só podem se comunicar usando transmissão de ordem total, a ordem causal é satisfeita trivialmente simplesmente aplicando a ordem FIFO.

contraste claro com os outros algoritmos da mesma classe, onde as mensagens são transmitidas diretamente para os destinos. O protocolo Train é, portanto, menos atraente do que os outros em uma rede de transmissão.

7.3.3. Token-FD. Ekwall et ai. [2004] também apresentam um algoritmo baseado na passagem de tokens em um anel. O algoritmo é especial porque depende de um detector de falha não confiável para tolerar falhas, enquanto todos os outros algoritmos baseados em token usam uma forma de associação de grupo.

Na versão básica do algoritmo, o token é o único portador de informação, assim como no protocolo Train (também é descrita uma otimização, na qual o token carrega identificadores de mensagem). No entanto, o token é enviado não apenas para o sucessor imediato no anel, mas para *f* +1 sucessores, onde *f* é o número de travamentos que o algoritmo tolera. As cópias adicionais são usadas apenas se um processo suspeitar da falha de seu predecessor. Para sua vivacidade, o algoritmo requer um detector de falha definido especificamente para anéis.

Este detector de falha é mais forte que S, mas mais fraco que P (consulte a Seção 5.3.1).

7.3.4. Totem. A especificidade do Totem [Amir et al. 1995] em comparação com outros algoritmos baseados em privilégios é que ele é projetado para sistemas particionáveis. A garantia de ordem assegurada é o Pedido Total Forte. O totem fornece acordo *(não uniforme)* e ordem total (chamada *ordem acordada)* e concordância *uniforme* e ordem total (chamada *ordem segura),* quando operado em um sistema não particionável. A ordem causal também é assegurada.

O algoritmo usa um protocolo de associação, que tem a responsabilidade de detectar falhas do processador, particionamento de rede e perda do token. Quando essas falhas são detectadas, o protocolo de associação reconstrói um novo anel, gera um novo token e recupera as mensagens que não foram recebidas por alguns dos processadores quando ocorreu a falha.

Os autores observam que, ao mover algoritmos sequenciadores (nos quais não é necessário segurar o token para ampliar transmitir uma mensagem) têm boa latência em cargas baixas, aumentos de latência em cargas altas e na presença de travamentos do processador. Mais ainda, de acordo com Agarwal et al. [1998], o anel e o esquema de passagem de token tornam os algoritmos baseados em privilégios altamente eficientes em LANs broadcast, mas menos adequados para LANs interconectadas. Para superar esse problema, eles estendem o Totem para um ambiente que consiste em várias LANs interconectadas. O algoritmo resultante funciona melhor em tal ambiente, mas, fora isso, tem as mesmas propriedades que o original de anel único.

7.3.5. TPM. O TPM [Rajagopalan e McKinley 1989] está intimamente relacionado ao Totem. A principal diferença é que o TPM não é particionável (só suporta associação à partição primária). Além disso, o TPM fornece apenas um acordo *uniforme* e um pedido total. Por fim, embora o TPM ofereça suporte apenas a um grupo fechado, os autores discutem algumas ideias sobre como estender o algoritmo para oferecer suporte a vários grupos fechados.

Rajagopalan e McKinley [1989] também propõem uma modificação do TPM em que as requisições de retransmissão são enviadas separadamente do token para melhorar o comportamento em redes com alta taxa de perda de mensagens.

7.3.6. Gopal e Toueg. O algoritmo de Gopal e Toueg [1989] é baseado no modelo round síncrono. O modelo round síncrono é um modelo de computação no qual a execução dos processos é sincronizada de acordo com os rounds. Durante cada rodada, cada processo executa as mesmas ações: (1) envia uma mensagem para todos os processos, (2) recebe uma mensagem de todos os processos não interrompidos e, então, (3) realiza alguns cálculos.

O algoritmo funciona da seguinte maneira. Para cada rodada, um dos processos é designado como *transmissor.* O transmissor de alguma rodada *r* é o único processo que tem permissão para transmitir novas mensagens de aplicação na rodada r. Nessa rodada, os outros processos transmitem confirmações de mensagens anteriores. mensagens

são entregues assim que são reconhecidos, três rodadas após sua transmissão inicial.

7.3.7. RTCAST. RTCAST [Abdelzaher et al. 1996] foi projetado para aplicações que precisam de garantias em tempo real.
O algoritmo assume um sistema síncrono com relógios sincronizados. Essas fortes garantias permitem a simplificação do protocolo. O trabalho também mostra como o tempo máximo de rotação do token pode ser utilizado para o controle de admissão e análise de escalonamento de mensagens em tempo real (com o objetivo de garantir o prazo de entrega dessas mensagens).

7.3.8. MARTE. MARS [Kopetz et al. 1991] é baseado na técnica de *acesso múltiplo por divisão de tempo* (TDMA; ver Nota 11). O TDMA consiste em intervalos de tempo periódicos predeterminados atribuídos a cada processo. Os processos podem então enviar ou difundir mensagens somente durante seus próprios intervalos de tempo. O sistema assume que os processos têm relógios sincronizados, pelo que são capazes de determinar com precisão o início e o fim do seu próprio intervalo de tempo. Além disso, supõe-se que a comunicação seja confiável e com atrasos limitados.

Com base na exclusão mútua fornecida pelo TDMA e no modelo de comunicação, a transmissão de ordem total é facilmente implementada. O mecanismo de ordenação pode ser visto como similar ao algoritmo de Gopal e Toueg (Seção 7.3.6), mas em um modelo baseado no tempo, onde a comunicação usa o tempo ao invés de mensagens [Lamport 1984].

Kopetz et ai. [1991] não discutem o comportamento de seu algoritmo de transmissão de ordem total na presença de falhas. Isso torna difícil determinar se o algoritmo é uniforme ou não. Acreditamos que não seja uniforme, simplesmente porque a uniformidade induz um custo de desempenho que os autores dificilmente considerariam acessíveis.

7.4. Algoritmos de Histórico de Comunicação

7.4.1. Lamport. O princípio do algoritmo de Lamport [Lamport 1978b], que usa relógios lógicos, foi explicado na Seção 4.4 (ver Figura 11). Na verdade, o artigo descreve um algoritmo de exclusão mútua. No entanto, é direto derivar um algoritmo de transmissão de ordem total a partir do algoritmo de exclusão mútua.

Como a ordem de entrega de uma mensagem *m* é determinada pelo timestamp do evento de transmissão de m, a ordem total é uma extensão da ordem causal. O algoritmo não é tolerante a falhas.

Um algoritmo semelhante é descrito por Attiya e Welch [1994], ao comparar critérios de consistência.

7.4.2. Psinc. O algoritmo Psync [Peterson et al. 1989] é usado em vários sistemas de comunicação de grupo: Consul [Mishra et al. 1993], Coyote [Bhatti et al. 1998], e Cactus [Hiltunen et al. 1999].

No Psync, os processos constroem dinamicamente um gráfico de causalidade das mensagens que recebem. Psync então entrega mensagens de acordo com uma ordem total que é uma extensão da ordem causal.

Psync assume um modelo de sistema assíncrono com falhas de travamento (permanentes) e comunicação com perdas (transitórias).
Para tolerar falhas de processo, o algoritmo parece assumir um detector de falhas perfeito, embora isso não seja declarado explicitamente no artigo. Para implementar canais confiáveis, o algoritmo usa reconhecimentos negativos (para solicitar a retransmissão de mensagens perdidas).

Psync é especificado apenas informalmente. No entanto, acreditamos que o protocolo garante a Ordem Total na ausência de falhas. Infelizmente, o comportamento diante de falhas não é descrito com detalhes suficientes para fazer uma afirmação segura sobre isso.
Acordo é um pouco mais complexo. Na ausência de perda de mensagem, a Psyc garante o Acordo. No entanto, com certas combinações de travamento do processo e perda de mensagem, é possível que alguns processos corretos *descartem* mensagens que seriam entregues por outros. Portanto, quando a perda da mensagem é considerada, o acordo pode ser violado. Esse problema é discutido em detalhes pelos autores, que o relacionam a uma instância do "último problema de reconhecimento".

Malhis et ai. [1996] fornecem uma análise do desempenho do Psync na presença de perda de mensagem. Eles concluíram que o Psync funciona bem se os broadcasts forem frequentes e a perda de mensagens for rara, mas funciona mal quando os broadcasts forem infrequentes e a perda de mensagens for comum. Eles mostram que o desempenho pode ser melhorado enviando regularmente mensagens vazias, como é feito por outros algoritmos de histórico de comunicação (consulte a Nota 14 na Seção 4.4).

7.4.3. Newtop (simétrico). Ezhilchelvan et ai. [1995] propõem dois algoritmos: um simétrico e um assimétrico. O algoritmo simétrico estende o algoritmo de Lamport (Seção 7.4.1) de várias maneiras: torna-o tolerante a falhas, permite que um processo seja membro de vários grupos e permite a difusão de uma mensagem para vários grupos. Quanto ao algoritmo de Lamport, Newtop preserva a ordem causal.

O Newtop é baseado em um serviço de associação de grupo particionável (consulte a Seção 3.4). A plataforma Newtop deixa para os aplicativos decidirem se devem ou não manter mais de um subgrupo em tal situação. Newtop satisfaz a propriedade de Weak Total Order mencionada na Seção 3.4.

O algoritmo assimétrico pertence a uma classe diferente e, portanto, é discutido na seção relevante (Seção 7.6.1). Os dois algoritmos (simétrico e assimétrico) podem ser facilmente combinados para permitir o uso do algoritmo simétrico em alguns grupos e do algoritmo assimétrico em outros.

7.4.4. Ng. Ng [1991] apresenta um algoritmo de histórico de comunicação que usa uma árvore geradora de custo mínimo para propagar mensagens. A ordem das mensagens é baseada nos relógios de Lamport, semelhantes ao algoritmo de Lamport. No entanto, mensagens e confirmações são propagadas e reunidas, usando uma spanning tree de custo mínimo. O uso de uma spanning tree melhora a escalabilidade do algoritmo e o torna adequado para redes de longa distância.

7.4.5. ToTo. O algoritmo ToTo [Dolev et al. 1993] garante a Ordem Total Fraca (consulte a Seção 3.4; é chamada de “multidifusão combinada” em Dolev et al. [1993]). Ele é construído sobre o sistema de comunicação de grupo particionável Transis [Dolev e Malkhi 1996]. ToTo estende a ordem de um algoritmo de transmissão causal subjacente. Baseia-se na construção dinâmica de um gráfico de causalidade das mensagens recebidas. O sistema Transis oferece uma variante uniforme e não uniforme do algoritmo. Uma particularidade de ToTo (variante não uniforme) é que, para entregar uma mensagem m, um processo deve ter recebido confirmações para m, de apenas alguns processos da maioria dos processos na visão atual (em vez de todos os membros da visão).

7.4.6. Total. O algoritmo Total [Moser et al. 1993] é construído sobre um algoritmo de transmissão confiável chamado Trans (Trans é definido junto com Total). No entanto, Trans não é usado como uma caixa preta (o que explica por que não listamos transmissão confiável como bloco de construção para esse algoritmo na Tabela IV). Trans usa um mecanismo de reconhecimento que define uma ordem parcial nas mensagens. Total estende a ordem parcial de Trans em uma ordem total. Duas variantes são definidas: a mais eficiente tolera $f < n/3$ falhas e a outra tolera $f < n/2$ falhas.

O algoritmo Total cumpre a propriedade de Acordo (na verdade, Acordo Uniforme) com alta probabilidade. Na verdade, o Total requer o protocolo de transmissão confiável Trans subjacente para fornecer garantias probabilísticas sobre a não reordenação das mensagens. Isso tem algumas semelhanças com a noção de *oráculos de ordenação fracos* (consulte a Seção 7.5.13).

Moser e Melliar-Smith [1999] propõem uma extensão do Total para tolerar falhas bizantinas.

7.4.7. ATOP. ATOP [Chockler et al. 1998] é um algoritmo que segue a abordagem de mesclagem determinística (Seção 4.4). O foco do artigo é adaptar o algoritmo para taxas de envio diferentes e possivelmente variáveis. Um número pseudoaleatório

O gerador é usado no cálculo da ordem de entrega.

O papel está principalmente preocupado em garantir uma propriedade ordenada. Esta propriedade é a Ordem Total Forte, definida no contexto de sistemas particionáveis (Seção 3.4). O algoritmo garante a ordem FIFO e garante a ordem causal trivialmente (consulte a nota de rodapé 16).

7.4.8. COREL. O algoritmo COReL [Keidar e Dolev 2000] é construído sobre um serviço de associação de grupo particionável como o Transis. O serviço subjacente também deve oferecer Ordem Total Forte (Seção 3.4), bem como ordem causal. O COReL gradualmente constrói uma ordem global (Reliable Total Order) marcando as mensagens de acordo com três níveis de cores diferentes (vermelho, amarelo baixo, verde). Uma mensagem começa em vermelho (sem conhecimento de sua posição na ordem global), depois passa para amarelo (recebido e reconhecido quando o processo é membro de um componente majoritário) e verde (todos os membros do componente majoritário reconheceram a mensagem , e sua posição na ordem global é conhecida). As mensagens verdes são entregues ao aplicativo.
As mensagens são retransmitidas e promovidas para verde sempre que as partições são mescladas. Todas as confirmações enviadas pelo algoritmo são carregadas. O COReL fornece a seguinte garantia de vivacidade: se eventualmente houver um componente de maioria estável, todas as mensagens enviadas pelos membros desse componente são entregues.

O COReL também oferece suporte à recuperação de processos se os processos estiverem equipados com armazenamento estável. Isso requer que os processos registrem cada mensagem enviada (antes de enviar a mensagem) e cada mensagem recebida (antes de enviar uma confirmação).

Fekete et ai. [2001] formalizam uma variante do algoritmo COReL e as garantias oferecidas pelo serviço de associação de grupo subjacente, usando autômatos de I/O.

7.4.9. Mesclagem Determinística. A principal motivação para o algoritmo de merge determinístico de Aguilera e Strom [2000] é minimizar o tempo esperado que um mes O Sage é atrasado para garantir a ordem total e para ter o menor número possível de mensagens enviadas pelos processos de destino. O algoritmo é projetado para sistemas nos quais vários remetentes enviam um fluxo constante de mensagens (a uma taxa aproximadamente fixa). Neste algoritmo, cada mensagem recebida define de forma terministica o remetente da próxima mensagem a ser aceita. O algoritmo depende de relógios aproximadamente sincronizados que são usados pelos remetentes para colocar um carimbo de data/hora físico em suas mensagens. Ao receber tal mensagem com registro de data e hora, um processo de destino calcula (usando o registro de data e hora e as taxas de envio de mensagens) o próximo remetente do qual aceitará uma mensagem. A qualidade da sincronização é importante para garantir o bom desempenho do algoritmo, mas não é necessária para sua correção. O algoritmo é mais eficiente se os relógios estiverem sincronizados (mas funciona mesmo que não estejam) e cada remetente envia mensagens a uma taxa fixa conhecida a priori (a taxa pode ser diferente para cada remetente). Para garantir a vivacidade do algoritmo, os remetentes precisam enviar mensagens vazias quando não tiverem nenhuma mensagem para enviar (essas mensagens dividem a execução em *épocas independentes).* O algoritmo, conforme descrito, não é tolerante a falhas.

7.4.10. TEM. Cristiano e cols. [1995] propõem uma coleção de algoritmos de transmissão ampla de ordem total (chamados HAS) que assumem um modelo de sistema síncrono com relógios sincronizados. Os autores descrevem três algoritmos—HAS-O, HAS-T e HAS-B—que , são respectivamente tolerantes a falhas de omissão, falhas de temporização e falhas bizantinas. Esses algoritmos são baseados no princípio da *difusão de informações,* que é baseado na noção de flooding, ou fofoca. Resumindo, quando um processo deseja transmitir uma mensagem m, ele marca o tempo de emissão T, de acordo com seu relógio local, e a envia para todos os vizinhos. Sempre que um processo recebe *m* pela primeira vez, ele o retransmite para seus vizinhos. Os processos entregam, de acordo com a mensagem *m* no relógios locais (onde é tempo *T* + seus constante que

depende da topologia da rede, do número de falhas toleradas e do desvio máximo do clock).

O artigo prova que os três algoritmos HAS satisfazem a concordância. Os autores não provam a Ordem Total, mas pelas propriedades dos relógios sincronizados e dos timestamps, a Ordem Total Uniforme não é muito difícil de impor. No entanto, se as suposições síncronas não forem válidas, os algoritmos podem violar a segurança do protocolo (ou seja, Ordem Total), em vez de apenas sua vivacidade.

7.4.11. Canais de transmissão redundantes. Cristian [1990] apresenta uma adaptação do algoritmo HAS-O (falhas de omissão) para canais broadcast. O modelo do sistema assume a disponibilidade de *f* + 1 canais de transmissão independentes (ou redes) que conectam todos os processos juntos, criando assim *f* + 1 caminhos de comunicação independentes entre quaisquer dois processos (onde *f* é o número máximo de falhas). Comparado ao HAS-O, o algoritmo para canais de transmissão redundantes emite significativamente menos mensagens.

7.4.12. Quick-S. Berman e Bharali [1993] apresentam vários algoritmos de transmissão de ordem total em uma variedade de modelos de sistema. Em sistemas síncronos (três variantes no artigo), os algoritmos são semelhantes aos algoritmos HAS: as mensagens são marcadas com data e hora (com marcações físicas ou lógicas, dependendo do modelo do sistema) e uma mensagem, marcada com T, pode ser entregue em *T* + por algum valor de . A diferença é que eles usam um algoritmo de acordo þizantino com um tempo de término limitado para enviar mensagens. Existem algoritmos que funcionam com falhas bizantinas e outros que funcionam apenas com falhas de travamento; os últimos asseguram a Ordem Uniforme de Prefixos.

Para falhas bizantinas, o algoritmo garante apenas propriedades não uniformes. Isso porque, ao contrário da *especificação* da parte Ram (Seção 7.1.9), a *especificação* utilizada pelo Quick-S não distingue entre processos bizantinos e aqueles que só falham por travamento.

O artigo também apresenta um algoritmo para sistemas assíncronos. No entanto, este algoritmo pertence à classe de algoritmos de acordo de destinos e é discutido lá (Quick-A; Seção 7.5.14).

7.4.13. ABP. O princípio da ABP [Minet e Anceaume 1991b; An ceaume 1993a] está próximo do princípio do algoritmo de Lamport (Seção 7.4.1): as mensagens são entregues de acordo com os timestamps anexados às mensagens por seu remetente. Cada processo gerencia uma variável de número de sequência local, usada para mensagens de carimbo de data/hora. Deixe o processo *p* transmitir a mensagem m. Na primeira fase, *m* e seu valor timestamp *tsm* são enviados para todos os processos. Qualquer processo *q* que recebe a mensagem *m* envia de volta uma resposta para p. A resposta do processo *q* para *p* também pode incluir se *q* teve anteriormente alguma mensagem mesmo valor de , *m* transmitida *m* com o timestamp *(tsm = tsm).* Ao receber todas as respostas dos processos corretos, o processo *p* conhece o conjunto *Msg(tsm)* de todas as mensagens com o mesmo valor de timestamp *tsm.* O processo *p* entrega essas mensagens (ordenadas de acordo com o identificador do remetente de cada mensagem). O processo *p* também transmite o conjunto *Msg(tsm),* permitindo assim que os outros processos entreguem a mesma sequência de mensagens.

7.4.14. Átomo. Em Atom [Bar-Joseph et al. 2002], fluxos de mensagens de todos os remetentes são mesclados de forma round-robin. Para tornar os algoritmos ativos, os remetentes precisam enviar mensagens vazias se não tiverem nenhuma mensagem para enviar. Essa abordagem pode ser vista como um caso especial de *fusão determinística* (consulte a Seção 7.4.9).

7.4.15. QoS preservando transmissão atômica. Bar-Joseph et al. [2000] apresentam outro algoritmo, baseado no mesmo mecanismo de ordenação do Atom (Seção 7.4.14). Como o próprio nome indica, o QoS preservando o algoritmo fornece suporte para qualidade de serviço (QoS), ao contrário do Atom. Por outro lado, o algoritmo de preservação de QoS não garante Concordância (ou seja, uniforme ou não), e apenas Ordem Total não uniforme.

7.5. Algoritmos de Contrato de Destinos

7.5.1. Skeen. O algoritmo de Skeen, descrito por Birman e Joseph [1987], foi usado em uma versão inicial do kit de ferramentas Isis.
O algoritmo corresponde aproximadamente ao algoritmo da Figura 14. A principal diferença é que o algoritmo de Skeen calcula o timestamp global de maneira centralizada, enquanto o algoritmo da Figura 14 o faz de maneira descentralizada.
A tolerância a falhas é alcançada usando um serviço de associação de grupo, que exclui processos suspeitos do grupo.

Dasser [1992] propõe uma otimização simples do algoritmo de Skeen chamada TOMP, onde informações adicionais são anexadas às mensagens do protocolo para entregar as mensagens do aplicativo um pouco antes.

7.5.2. Luan e Gligor. Luan e Gligor [1990] propuseram um algoritmo baseado na votação por maioria. A ideia é a seguinte.
Após a execução de *TO-broadcast(m),* a mensagem *m* é enviada para todos os processos. Após a recepção de *m* por algum processo q, *m* é colocado no buffer de recepção de q. A ordem de entrega da mensagem é então decidida por um protocolo de votação, que pode ser iniciado por qualquer um dos processos. No caso de início simultâneo do protocolo, uma regra de arbitragem é usada.

A votação é iniciada pela transmissão de uma mensagem de “convite”. Considere esta mensagem transmitida pelo processo p. Os processos respondem enviando o conteúdo de seu buffer de recebimento para p. O processo *p* espera pela maioria das respostas. Com base nas mensagens recebidas, o processo *p* então constrói uma seqüência de identificadores de mensagem e transmite essa seqüência. Um processo que recebe a sequência envia uma confirmação para p. Uma vez que *p* tenha recebido confirmações da maioria dos processos, a sequência proposta é *confirmada.*

Resumindo, o protocolo tenta chegar a um consenso entre os processos de destino em uma sequência de mensagens. No entanto, os autores não identificaram o consenso como um subproblema a resolver, o que torna o protocolo mais complexo. A consequência é que as condições sob as quais vivacidade é garantida não são discutidas (e difíceis de inferir).

7.5.3. Le Lann e Bres. Le Lann e Bres [1991] escreveram um documento de posição discutindo a transmissão de ordem total em um sistema com falhas de omissão. O artigo esboça um algoritmo de transmissão de ordem total baseado em quóruns.

7.5.4. Chandra e Toueg. Chandra e Toueg [1996] propõem uma transformação da transmissão atômica em uma sequência de problemas de consenso, onde cada consenso decide sobre um conjunto de mensagens, facilmente transformadas em uma sequência de mensagens. A transformação usa transmissão ampla confiável. A ideia dessa transformação, descrita na Seção 4.5, não é repetida aqui.

O algoritmo assume um modelo de sistema assíncrono, transmissão confiável e uma caixa preta que resolve o consenso.
O algoritmo é extremamente elegante, no sentido de que todos os problemas difíceis relacionados à tolerância a falhas estão ocultos na caixa preta consensual.

Houve várias propostas para otimizar esse algoritmo. Por exemplo, Mostefaoui e Raynal [2000] propõem uma abordagem otimista na qual o algoritmo de consenso é dividido em duas partes. A primeira fase é otimizada, mas nem sempre é bem-sucedida. Se isso acontecer, o algoritmo de consenso completo é executado.

7.5.5. Rodrigues e Raynal. Rodrigues e Raynal [2000] apresentam um algoritmo de broadcast de ordem total em um modelo onde os processos têm acesso a armazenamento estável e podem se recuperar após uma falha. O algoritmo é muito próximo do algoritmo de Chandra-Toueg (Seção 7.5.4): ele usa a mesma transformação de ordem total transmitida para consenso. A única diferença é que, por causa do modelo de recuperação de falhas, o algoritmo depende do algoritmo de consenso de recuperação de falhas de Aguilera, Chen e Toueg [2000]).

7.5.6. ATR. Delporte-Gallet e Fau connier [1999] descrevem o algoritmo ATR, que é baseado em uma abstração

chamado *Sistema de Fase Sincronizada (SPS).*
A abstração SPS é definida em um sistema assíncrono. Um SPS decompõe a execução de um algoritmo em rodadas, quase como um modelo de rodada síncrona.
O algoritmo ATR distingue entre rodadas pares e ímpares. Em rodadas pares, os processos enviam conjuntos ordenados de mensagens entre si. Ao receber essas mensagens, cada processo constrói uma sequência de mensagens. Na rodada ímpar subsequente, os processos tentam validar o pedido e enviar mensagens.

7.5.7. SCALATOM. SCALATOM Rodrigues et al. [1998] é baseado no algoritmo de Skeen (Seção 7.5.1) e suporta o broadcast de mensagens para múltiplos grupos.
O algoritmo satisfaz a propriedade de Minimalidade Forte (Seção 3.2.4). O timestamp global é calculado usando uma variante do algoritmo de consenso de Chandra e Toueg [1996] (Seção 7.5.4). SCALATOM corrige um algoritmo anterior, chamado MTO [Guerraoui e Schiper 1997].

7.5.8. Fritzke et ai. Fritzke et ai. [2001] também propõem um algoritmo para o broadcast de mensagens para múltiplos grupos. O algoritmo satisfaz a propriedade Strong Minimality (Seção 3.2.4). Considere uma mensagem *transmitida* para vários grupos. Primeiro, o algoritmo usa o consenso para decidir sobre o timestamp de *m* dentro de cada grupo de destino. Os grupos de destino então trocam informações para calcular o timestamp final, e um segundo consenso é executado em cada grupo para atualizar o relógio lógico.

7.5.9. Transmissão atômica otimista. Otimismo é uma técnica utilizada há vários anos no contexto de controle de concorrência [Bernstein et al. 1987] e replicação do sistema de arquivos [Guy et al. 1993]. No entanto, só recentemente foi considerado no contexto da transmissão de ordem total [Pedone 2001].

O algoritmo de transmissão atômica otimista de Pedone e Schiper [1998, 2003] é baseado na observação experimental de que as mensagens transmitidas em uma LAN são geralmente todos recebidos na mesma ordem por todos os processos. Quando essa suposição é atendida, o algoritmo entrega as mensagens muito rapidamente. No entanto, se a suposição não for válida, o algoritmo é menos eficiente do que outros algoritmos (mas ainda entrega as mensagens na ordem total).

Ao contrário da maioria dos algoritmos otimistas, a transmissão atômica otimista de Pedone e Schiper [2003] é otimista *internamente.*
Isso significa que o mecanismo otimista do algoritmo não é aparente para o aplicativo. Em outras palavras, não há enfraquecimento das propriedades de entrega.

7.5.10. Prefix Acordo. Antecedentes [1997] define uma variante de consenso, chamada de *acordo de prefixo,* onde os processos concordam em um fluxo de valores, ao invés de um único valor. A consideração de fluxos em vez de valores únicos torna o algoritmo de acordo de prefixo particularmente adequado para resolver a ordem total de transmissão.

O algoritmo de transmissão de ordem total usa acordo de prefixo para decidir repetidamente sobre a sequência de mensagens a serem entregues a seguir.

7.5.11. Transmissão Genérica. Elenco amplo genérico [Pedone e Schiper 1999; 2002] não é uma transmissão de ordem total per se. Em vez disso, o algoritmo assume uma relação *de conflito* nas mensagens, e duas mensagens *m* e *m* são entregues na mesma ordem em cada processo de destino, somente se forem conflitantes. Duas mensagens *m* e *m* que não são conflitantes não são ordenadas pelo algoritmo. Se todas as mensagens forem conflitantes, a transmissão genérica oferece a mesma garantia que a transmissão de ordem total. Se nenhuma mensagem for conflitante, então a transmissão genérica fornece as garantias de transmissão (uniforme) confiável. O ponto forte desse algoritmo é que o desempenho varia de acordo com a "quantidade de ou dering" necessária. O algoritmo de transmissão genérico usa um algoritmo de consenso apenas em caso de conflitos.

7.5.12. Transmissão genérica econômica. Aguil foi, Delporte-Gallet et al. [2000] também

propor um algoritmo de transmissão genérico. Quando mensagens conflitantes são detectadas, Pedone e Schiper [2002] resolvem a transmissão genérica por redução ao consenso, enquanto Aguilera et al. [2000] resolve a transmissão genérica por redução à transmissão de ordem total. Além disso, o algoritmo é *econômico* no sentido de que, se houver um tempo após o qual as mensagens de transmissão não entrem em conflito umas com as outras, então, eventualmente, a difusão atômica não será mais usada. O algoritmo de Pedone e Schiper [2002] também satisfaz esta propriedade com relação ao consenso, mas a propriedade não foi identificada no artigo.

7.5.13. Oráculos de Ordenação Fracos. Pedone et ai. [2002] definem um *oráculo de ordenação fraco* como um oráculo que ordena as mensagens que são transmitidas, mas pode cometer erros (ou seja, as mensagens transmitidas podem ser entregues fora de ordem). Este oráculo modela o comportamento observado em redes locais, onde as mensagens de broadcast são entregues espontaneamente em ordem total.
O artigo mostra que a ordem total ampla pode ser resolvida usando um oráculo de ordem fraca. Se a suposição otimista for atendida, o algoritmo proposto, que assume $f < \frac{n}{3}$, resolve a ordem total transmitida em duas etapas de comunicação.

Curiosamente, o algoritmo tem a mesma estrutura do algoritmo de consenso aleatório proposto por Rabin [1983].
Os autores também mencionam que o oráculo de ordem fraca poderia ser usado para projetar um algoritmo de transmissão de ordem total com a mesma estrutura do algoritmo de consenso aleatório proposto por Ben-Or [1983].

7.5.14. Quick-A. Berman e Bharali [1993] apresentam uma série de quatro algoritmos, três dos quais pertencem a outra classe (ver Quick-S, Seção 7.4.12). Seu algoritmo para sistemas assíncronos é bastante diferente de seus algoritmos para sistemas síncronos (Seção 7.4.12). Os processos mantêm um número redondo e as mensagens de transmissão são marcadas com esse número redondo. Os processos então executam uma sequência de consenso binário aleatório, para decidir sobre a rodada em que as mensagens devem ser entregues.

7.5.15. AMp /xAMP. O AMp [Ver´ÿssimo et al. 1989] e xAMp [Rodrigues e Ver´ÿssimo 1992] partem da suposiç˜ao de que, na maioria das vezes, as mensagens broadcast s˜ao recebidas por todos os processos de destino na mesma ordem (uma suposiç˜ao realista em LANs, como já mencionado).
Assim, quando um processo transmite uma mensagem, ele inicia um protocolo de compromisso. Se as mensagens forem recebidas em ordem por todos os processos de destino, o resultado é positivo: todos os processos de destino confirmam e entregam a mensagem. Caso contrário, a mensagem é rejeitada e o remetente deve tentar novamente (possivelmente levando a um livelock).

7.6. Algoritmos Híbridos

Aqui discutimos algoritmos que não se encaixam em nenhuma de nossas cinco classes de algoritmos de transmissão de ordem total. Esses algoritmos geralmente combinam dois mecanismos diferentes ou diferentes.

7.6.1. Newtop (assimétrico). Ezhilchel van et ai. [1995] propõe dois algoritmos; um simétrico e outro assimétrico.
O algoritmo simétrico foi descrito anteriormente (Seção 7.4.3).

O algoritmo assimétrico usa um processo sequenciador e permite que um processo seja membro de vários grupos (cada grupo possui um sequenciador independente). Para a ordenação, o algoritmo usa os relógios lógicos de Lamport [1978b] além do sequenciador. Portanto, o algoritmo assimétrico é um híbrido entre um algoritmo de histórico de comunicação (devido ao uso dos relógios de Lamport) e um algoritmo de sequenciador fixo. O algoritmo assimétrico, como o simétrico, preserva a entrega causal ou der. No entanto, observe que um processo p, que é membro de mais de um grupo, não pode transmitir uma mensagem *m* para um grupo imediatamente após o broadcast de alguma mensagem *m* para um grupo diferente. O processo *p* só pode entregar *m* depois de ter entregado m. Portanto, o algoritmo assimétrico não permite tecnicamente uma

mensagem seja transmitida para mais de um grupo.

Conforme mencionado na Seção 7.4.3, Newtop [Ezhilchelvan et al. 1995] suporta a combinação de grupos, mesmo que um grupo use os algoritmos assimétricos e o outro grupo use o simétrico. Além disso, o Newtop é baseado em um serviço de associação de grupo particionável.

7.6.2. Híbrido. Rodrigues e cols. [1996] apresentam um algoritmo otimizado para grandes redes. O algoritmo é híbrido: em escala local, um número de sequência é anexado a cada mensagem por um sequenciador fixo, e em escala global, a ordenação é do tipo histórico de comunicação. Mais precisamente, cada remetente *p* tem um processo sequenciador associado que emite um número de sequência para cada mensagem de p. A mensagem original e seu número de sequência são enviados a todos, e as mensagens são finalmente ordenadas usando uma técnica de histórico de comunicação padrão (consulte a Seção 7.4.1). Os autores também descrevem heurísticas interessantes para alterar o processo do sequenciador. As razões para tais alterações podem ser falhas, alterações de associação ou alterações no padrão de tráfego.

7.6.3. Ordem Total Uniforme Indulgente. Vicente e Rodrigues [2002] propõem um algoritmo otimista para redes de longa distância. O algoritmo é baseado no otimismo externo, conforme proposto inicialmente por Kemme et al. [1999, 2003]. Isso significa que o algoritmo distingue entre dois eventos de entrega seguindo a transmissão da mensagem m: a entrega otimista, denotada *Opt-deliver(m),* e a entrega tradicional de pedido total, denotada *Adeliver(m).*

Após *Opt-deliver(m),* a ordem de entrega de *m* ainda não foi decidida. No entanto, o aplicativo pode começar a processar m. Se posteriormente, *To-deliver(m)* invalidar a ordem de entrega otimista, então o aplicativo deve reverter e desfazer o processamento de m. O otimismo de Kemme et al. [2003] está relacionado ao ordenamento total espontâneo em LANs.

O algoritmo otimista de Vicente e Rodrigues [2002] estende o algoritmo híbrido de Rodrigues et al. [1996] (Seção 7.6.2). A ordem de entrega é determinada por números de sequência anexados às mensagens. Um número de sequência anexado a uma mensagem *m* deve ser validado por uma maioria de processos antes que a ordem total de *m* seja decidida. No entanto, o algoritmo entrega *m* de forma otimista de acordo com seu número de sequência antes que o número de sequência seja realmente validado.

7.6.4. Ordem Total Otimista em WANs. Algoritmos de transmissão de ordem total otimistas dependem fortemente da suposição de que as mensagens são muitas vezes recebidas por todos os processos em alguma ordem total espontânea. Essa suposição foi motivada por observações feitas em redes locais, geralmente em um único hub. A suposição é, no entanto, questionável para redes de longa distância, nas quais a ordem total espontânea é significativamente menos provável de ocorrer. Sousa et ai. [2002] propõem uma solução baseada em tempo para resolver esse problema e aumentar a probabilidade de ordem total espontânea em redes de longa distância. A técnica, chamada de *compensação de atraso,* consiste em atrasar artificialmente as mensagens recebidas, de modo que todos os destinos as processem aproximadamente ao mesmo tempo. Um atraso é mantido para cada canal de comunicação de entrada e a duração desse atraso é adaptada dinamicamente.

8. OUTRO TRABALHO SOBRE A ORDEM TOTAL E QUESTÕES RELACIONADAS

Além dos artigos que propõem algoritmos de transmissão de ordem total, há outros trabalhos intimamente relacionados que vale a pena mencionar.

Protocolo de recuo. Chockler, Malkhi e Reiter [2001] descrevem um protocolo de replicação que emula a *replicação da máquina de estado* [Lamport 1978a; Schneider 1990]. O protocolo é baseado em sistemas de quorum e depende de um protocolo de exclusão mútua. Basicamente, um processo cliente que deseja executar alguma operação *nos* servidores replicados procede da seguinte forma: o cliente primeiro espera para entrar na seção crítica e, em seguida, (1) acessa um quorum de réplicas para obter um estado atualizado ÿ dos servidores replicados, (2) executa a operação *op* em ÿ que leva a um novo estado ÿ ,

e (3) atualiza um quorum de réplicas com o novo estado ÿ . O protocolo é seguro mesmo se o protocolo de exclusão mútua violar a segurança (mais de um processo na seção crítica): a segurança do protocolo de exclusão mútua só é necessária para garantir o progresso do protocolo de replicação.

Replicação Ativa Otimista. Felber e Schiper [2001] descrevem outro protocolo de replicação que é integrado a um algoritmo de transmissão de ordem total. O protocolo de replicação é baseado em um algoritmo de difusão de ordem total de sequenciador fixo otimista, que é executado entre os servidores. O algoritmo otimista pode levar alguns servidores a entregarem mensagens fora de ordem, caso em que esses servidores precisam reverter. A reversão é limitada aos servidores; os processos do cliente nunca precisam ser revertidos.

Probabilístico Protocolos. Recentemente, Felber e Pedone [2002] propuseram um algoritmo de broadcast totalmente ordenado com segurança probabilística. Isso significa que seus algoritmos impõem as propriedades de transmissão de ordem total com uma probabilidade configurável. Isso abre espaço para soluções extremamente escaláveis, mas é aceitável apenas para aplicativos com requisitos muito fracos. Em particular, Felber e Pedone [2002] propõem uma especificação em que a concordância é garantida com a probabilidade *ÿa,* a ordem total com a probabilidade *ÿo* e a validade com a probabilidade *ÿv.* Os autores propõem um algoritmo baseado em fofocas e discutem condições suficientes sob as quais seu algoritmo pode impor as propriedades acima com probabilidade um.

Protocolos baseados em hardware. Devido à sua especificidade, omitimos deliberadamente algoritmos que fazem uso explícito de hardware dedicado. No entanto, eles merecem ser citados aqui. Alguns protocolos são baseados em uma modificação dos controladores de rede (por exemplo, Jalote [1998] e Minet e Anceaume [1991a]). A ideia é modificar um pouco a rede para que ela possa ser usada como um sequenciador virtual. Em nosso sistema de classificação, esses protocolos podem ser classificados como protocolos de sequenciador fixo. Alguns outros protocolos dependem das características de redes específicas, como uma topologia específica [Cordova ´ e Lee 1996], ou a capacidade de reservar buffers [Chen et al. 1996].

Desempenho dos Algoritmos de Transmissão de Ordem Total. Em comparação com o grande número de publicações que descrevem algoritmos, relativamente poucos artigos se preocupam com a avaliação do desempenho da transmissão total ou der (por exemplo, Cristian et al. [1994], Friedman e van Renesse [1997], Mayer [1992], descrito em Seção 1). Recentemente, apresentamos uma análise comparativa de desempenho com base na classificação desenvolvida nesta pesquisa [Defago et al. 2003]: algoritmos são retirados de todas as cinco classes de mecanismos de ordenação, e algoritmos uniformes e não uniformes são considerados. Urbano e outros. [2003] vão além de simplesmente avaliar algum algoritmo ou comparar diferentes algoritmos. Eles propõem benchmarks, incluindo métricas de desempenho bem definidas, cargas de trabalho e *cargas de falhas ,* descrevendo como falhas e eventos relacionados devem ocorrer.

Métodos formais. Métodos formais têm sido aplicados ao problema de transmissão de ordem total, a fim de provar as propriedades dos algoritmos [Zhou e Hooman 1995; Toinard et al. 1999; Fekete et ai. 2001], e ao problema do consenso, a fim de construir uma prova verdadeiramente formal para um algoritmo [Nestmann et al. 2003]. As provas de Fekete et al. [2001] foram posteriormente verificados por um provador de teorema. Liu et ai. [2001] usam a noção de meta-propriedades para descrever e analisar um protocolo que alterna entre dois algoritmos de transmissão de ordem total.

Controvérsia de Comunicação de Grupo. Onze anos atrás, Cheriton e Skeen [1993] publicaram uma polêmica sobre sistemas de comunicação de grupo que fornecem primitivas de comunicação totalmente ordenadas e causalmente. Seu principal argumento contra os sistemas de comunicação em grupo era que os sistemas baseados em transações são mais eficientes, ao mesmo tempo em que fornecem um modelo de consistência mais forte. Isso foi posteriormente respondido por Birman [1994] e Shrivastava [1994]. Mais de uma década depois, parece que o trabalho em sistemas de transações e em sistemas de comunicação de grupo tendiam a se influenciar mutuamente para benefício mútuo [Schiper

e Raynal 1996; Agrawal et ai. 1997; Pedone et ai. 1998; Kemme e Alonso 2000; Wiesmann et ai. 2000; Kemme et al. 2003].

9. CONCLUSÃO

A vasta literatura sobre difusão de ordem total e o grande número de algoritmos publicados mostram a complexidade do problema. No entanto, essa abundância de informações é um problema em si, pois dificulta a compreensão das compensações exatas associadas a cada solução proposta.

A principal contribuição deste artigo é a definição de uma classificação para algoritmos de broadcast de ordem total, que facilite o entendimento da relação entre eles. Ele também fornece uma boa base para comparar os algoritmos e entender algumas compensações. Além disso, o artigo apresentou um vasto levantamento da maioria dos algoritmos existentes e discutiu suas respectivas características.

Apesar do grande número de algoritmos de transmissão total ou der publicados, a maioria são apenas melhorias ou variantes uns dos outros (mesmo que isso não seja imediatamente óbvio para o olho destreinado). Na verdade, existem apenas alguns algoritmos verdadeiramente originais, mas uma grande coleção de várias otimizações. No entanto, é importante enfatizar que otimizações inteligentes de algoritmos existentes geralmente têm um impacto muito significativo no desempenho. Por exemplo, Friedman e van Renesse [1997] mostram que as mensagens de carona, apesar de sua simplicidade, podem melhorar significativamente o desempenho dos algoritmos.

Embora a especificação do problema de broadcast de ordem total remonte a algumas das primeiras publicações sobre o assunto, poucos artigos realmente especificam o problema que eles resolvem. Na verdade, muito poucos algoritmos são especificados adequadamente, muito menos comprovados como corretos. Felizmente, isso está mudando e esperamos que este artigo contribua para um trabalho mais rigoroso no futuro. Sem levar o formalismo ao extremo, uma especificação clara e uma prova sólida de correção são tão importantes quanto o próprio algoritmo. Com efeito, eles claramente definir os limites dentro dos quais o algoritmo pode ser usado.

AGRADECIMENTOS

Agradecemos às seguintes pessoas por seus conselhos úteis e críticas construtivas: David E. Bakken, Bernadette Charron-Bost, Adel Cherif, Alan D. Fekete, Takuya Katayama, Tohru Kikuno, Dahlia Malkhi, Keith Marzullo, Rui C. Oliveira, Fernando Pedone, Michel Raynal, Luis T. Rodrigues, Richard D. Schlichting, Tatsuhiro Tsuchiya , e Matthias Wiesmann , bem como Fred B. Schneider , e os revisores anônimos. Agradecemos também a Judith Steeh pela revisão deste texto.

REFERÊNCIAS

ABDELZAHER, T., SHAIKH, A., JAHANIAN, F., E SHIN, K. 1996. RTCAST: Multicast leve para grupos de processos em tempo real. No *IEEE Real-Time Tech nology and Applications Symposium (RTAS'96).* (Boston, MA). IEEE Computer Society Press. 250–259.

AGARWAL, DA, MOSER, LE, MELLIAR-SMITH, PM E BUDHIA, RK 1998. O protocolo de manutenção de topologia e ordenação de múltiplos anéis do Totem. *ACM Trans. Comput. Sist. 16,* 2 (maio), 93-132.

AGRAWAL, D., ALONSO, G., EL ABBADI, A. E STANOI, I. 1997. Explorando transmissão atômica em bancos de dados replicados (resumo estendido). In *Proceedings of EuroPar'97* (Passau, Germany). Notas de aula em Ciência da Computação, vol. 1300. Springer Verlag. 496–503.

AGUILERA, MK, CHEN, W., E TOUEG, S. 1999. Usando o detector de falha de pulsação para comunicação confiável quiescente e consenso em redes particionáveis. *Teor. Comput. ciência 220,* 1 (junho), 3–30.

AGUILERA, MK, CHEN, W., E TOUEG, S. 2000. Em comunicação confiável quiescente. *SIAM J. Comput. 29,* 6, 2040–2073.

AGUILERA, MK, DELPORTE-GALLET, C., FAUCONNIER, H., E TOUEG, S. 2000. Thrifty generic broad cast. In *Proceedings of 14th International Symposium on Distributed Computing (DISC'00)* (Toledo, Espanha). M. Herlihy, Ed. Notas de aula em Ciência da Computação, vol. 1914. Springer-Verlag. 268–282.

AGUILERA, MK E STROM, RE 2000. Transmissão atômica eficiente usando fusão determinística. In *Proceedings of 19th ACM Symposium on Principles of Distributed Computing (PODC-19)* (Portland, OU). Imprensa ACM. 209–218.

AMIR , Y. , DOLEV , D. , KRAMER , S. , & MALKI , D. . 1992. Transis: Um subsistema de comunicação para alta disponibilidade. Nos *Anais do 22º Simpósio Internacional de Computação Tolerante a Falhas (FTCS-22)* (Boston, MA). IEEE Computer Society Press. 76–84.

AMIR, Y., MOSER, LE, MELLIAR-SMITH, PM, AGARWAL, DA E CIARFELLA, P. 1995. O protocolo de associação e ordenação de anel único do Totem. *ACM Trans. Comput. Sist. 13,* 4 (novembro), 311–342.

ANCEAUME, E. 1993a. Algoritmos de confiabilidade para sistemas distribuídos.Tese de doutorado, Universidade de Paris-sud (Paris XI), Paris, França.

ANCEAUME, E. 1993b. Uma comparação de protocolos de transmissão atômica tolerantes a falhas. In *Proceedings of 4th IEEE Computer Society Workshop on Future Trends in Distributed Computing Systems (FTDCS-4)* (Lisboa, Portugal). IEEE Computer Society Press. 166–172.

ANCEAUME, E. 1997. Uma solução leve para transmissão atômica uniforme para sistemas assíncronos. Nos *Anais do 27º Simpósio Internacional de Computação Tolerante a Falhas (FTCS-27)* (Seattle, WA). IEEE Computer Society Press. 292–301.

ANCEAUME, E. E MINET, P. 1992. Estudo de protocolos de difusão atômica. TR 1774, INRIA (outubro) Rocquencourt, França.

ARMSTRONG, S., FREIER, A., E MARZULLO, K. 1992. Protocolo de transporte multicast. RFC 1301, IETF (fevereiro)

ATTIYA, H. E WELCH, JL 1994. Consistência sequencial versus linearizabilidade. *ACM Trans. Com colocar. Sist. 12,* 2 (maio), 91–122.

BAR-JOSEPH, Z., KEIDAR, I., ANKER, T., E LYNCH, N. 2000. QoS preservando multicast totalmente ordenado. In *Proceedings of 4th International Conference on Principles of Distributed Systems (OPODIS).* Studia Informatica, Paris, França, 143–162.

BAR-JOSEPH, Z., KEIDAR, I., E LYNCH, N. 2002. Transmissão atômica dinâmica de entrega antecipada. In *Actas do 16º Simpósio Internacional de Computação Distribuída (DISC'02)* (Toulouse, França). D. Malkhi, Ed. Notas de Palestra em Ciência da Computação, vol. 2508. Springer-Verlag. 1–16.

BEN-OR, M. 1983. Outra vantagem da livre escolha: Protocolos de acordo completamente assíncronos. In *Proceedings of 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC-2)* (Montreal Quebec, Canadá). Imprensa ACM. 27–30.

BERMAN, P. E BHARALI, AA 1993. Transmissão atômica rápida (resumo estendido). In *Pro ceedings of 7th International Workshop on Distributed Algorithms (WDAG'93)* (Lausanne, Suíça). A. Schiper Ed. Notas de aula em Ciência da Computação, vol. 725. Springer-Verlag. 189–203.

BERNSTEIN, PA, HADZILACOS, V., E GOODMAN, N. 1987. *Controle de Simultaneidade e Recuperação em Sistemas de Banco de Dados.* Addison-Wesley, Boston, MA. http://research.microsoft.com/pubs/ccontrol/.

BHATTI, NT, HILTUNEN, MA, SCHLICHTING, RD, E CHIU, W. 1998. Coyote: Um sistema para construir serviços de comunicação configuráveis de granulação fina. *ACM Trans. Comput. Sist. 16,* 4 (novembro), 321–366.

BIRMAN, K. 1994. Uma resposta à crítica de Cheriton e Skeen à comunicação causal e totalmente ordenada. *Operador ACM Sist. Rev. 28,* 1 (janeiro), 11–21.

BIRMAN, K. E VAN RENESSE, R. 1994. *Computação distribuída confiável com o kit de ferramentas Isis* (Los Alamitos, CA). IEEE Computer Society Press.

BIRMAN, KP E JOSEPH, TA 1987. Comunicação confiável na presença de falhas. *ACM Trans. Comput. Sist.* 5,1 (fevereiro), 47–76.

BIRMAN, KP, SCHIPER, A. E STEPHENSON, P. 1991. Multicast de grupos atômicos e causais leves. *ACM Trans. Comput. Sist.* 9, 3 (agosto), 272-314.

CARR, R. 1985. O protocolo de atualização global Tandem col. *Sistema Tandem Rev.* 1,2 (junho), 74–85.

CHANDRA , TD , HADZILACOS , V. , E TOUEG , S. 1996 . O detector de falha mais fraco para resolver o consenso. *J. ACM* 43,4 (julho), 685–722.

CHANDRA, TD E TOUEG, S. 1996. Detectores de falha não confiáveis para sistemas distribuídos confiáveis. *J. ACM 43,* 2, 225–267.

CHANG, J.-M. E MAXEMCHUK, NF 1984. Protocolos de transmissão confiáveis. *ACM Trans. Comput. Sist.* 2, 3 (agosto), 251–273.

CHARRON-BOST, B., PEDONE, F., E DEFAGO, X. 1999. Comunicações privadas. (Mostrou um exemplo ilustrando o fato de que mesmo a combinação de acordo forte, ordem total forte e integridade forte não impede que um processo defeituoso atinja um estado inconsistente.)

CHARRON-BOST, B., TOUEG, S., E BASU, A. 2000. Revisitando segurança e vivacidade no contexto de falhas. In *Concurrency Theory, 11ª Conferência Internacional (CONCUR 2000)* (University Park, PA). Notas de aula em Ciência da Computação, vol. 1877. Springer-Verlag. 552–565.

CHEN, X., MOSER, LE, E MELLIAR-SMITH, PM 1996. Multicasting totalmente ordenado baseado em reserva. In *Proceedings of 16th IEEE International Conference on Distributed Computing Systems (ICDCS-16)* (Hong Kong). IEEE Computer Society Press. 511–519.

CHERITON, DR E SKEEN, D. 1993. Compreender as limitações da comunicação causal e totalmente ordenada. Em *Anais do 14º Simpósio ACM sobre Princípios de Sistemas Operacionais (SoSP-14)* (Ashville, NC). Imprensa ACM. 44–57.

CHIU, G.-M. E HSIAO, C.-M. 1998. Uma nota sobre multicast de ordenação total usando árvores de propagação. *IEEE Trans. Paralelo. Dist. Sist.* 9, 2 (fevereiro), 217–223.

CHOCKLER, G., KEIDAR, I. E VITENBERG, R. 2001. Especificações de comunicação em grupo: um estudo abrangente. *Computação ACM. Sobreviver 33,* 4 (dez.), 427–469.

CHOCKLER, G.V., HULEIHEL, N., E DOLEV, D. 1998. Um protocolo multicast ordenado total adaptativo que tolera partições. Nos *Anais do 17º Simpósio ACM sobre Princípios*

de Computação Distribuída (PODC-17) (Puerto Vallarta, México). Imprensa ACM. 237–246.

CHOCKLER, GV, MALKHI, D., E REITER, MK 2001. Protocolos de backoff para exclusão e ordenação mútua distribuída. In *Proceedings of 21st IEEE International Conference on Distributed Com puting Systems (ICDCS-21)* (Phoenix, AZ). IEEE Computer Society Press. 11–20.

CHOR, B. E DWORK, C. 1989. Randomização no Acordo Bizantino. *Adv. Comput. Res.* 5, 443-497.

CORDOVA, J. E LEE, Y.-H. 1996. Árvores multicast para fornecer ordenação de mensagens em redes mesh. *Comput. Sist. ciência Eng.* 11,1 (janeiro), 3–13.

CRISTIAN, F. 1990. Transmissão atômica síncrona para canais de transmissão redundantes. *Sistema em tempo real.* 2, 3 (setembro), 195–212.

CRISTIAN, F. 1991. Elenco atômico assíncrono. *Boletim de Divulgação Técnica IBM 33,* 9, 115–116.

CRISTIAN, F., AGHILI, H., STRONG, R., DOLEV , D. 1995. Transmissão atômica: da difusão de mensagens simples ao acordo bizantino. *Inf. Com colocar. 18,* 1, 158–179.

CRISTIAN, F., DE BEIJER, R., E MISHRA, S. 1994. Uma comparação de desempenho de protocolos de transmissão atômica assíncrona. *Dist. Sist. Eng. J.* 1,4 (junho), 177–201.

CRISTIAN, F. E FETZER, C. 1999. O modelo de sistema distribuído assíncrono cronometrado. *IEEE Trans. Paralelo. Dist. Sist.* 10,6 (junho), 642–657.

CRISTIAN, F. E MISHRA, S. 1995. Os protocolos de transmissão atômica assíncrona cata-vento. In *Actas do 2º Simpósio Internacional de Sistemas Autônomos Descentralizados* (Phoenix, AZ). IEEE Computer Society Press. 215-221.

CRISTIAN, F., MISHRA, S., E ALVAREZ, G. 1997. Transmissão atômica assíncrona de alto desempenho. *Dist. Sist. Eng. J.* 4,2 (junho), 109–128.

DASSER, M. 1992. TOMP: Um protocolo multicast de ordenação total. *Operador ACM Sist. Rev. 26,* 1 (janeiro), 32–40.

DEFAGO, X. 2000. Problemas relacionados a acordos: Da replicação semi-passiva ao broadcast totalmente ordenado. doutorado tese, Ecole Polytechnique Fed erale de Lausanne, Lausanne, Suíça. Número 2229.

DEFAGO , X., SCHIPER, A., E URBAN P. 2003. , Análise comparativa de desempenho de estratégias de ordenação em algoritmos de transmissão atômica. *IEICE Trans. Inf. Sist. E86–D,* 12 (dez.), 2698–2709.

DELPORTE-GALLET, C. E FAUCONNIER, H. 1999. Transmissão atômica tolerante a falhas em tempo real. In *Proceedings of 18th IEEE International Symposium on Reliable Distributed Systems (SRDS'99)* (Lausanne, Suíça). IEEE Computer Society Press. 48–55.

DELPORTE-GALLET, C. E FAUCONNIER, H. 2000. Transmissão atômica genuína tolerante a falhas para mul grupos tipicos. In *Proceedings of 4th International Conference on Principles of Distributed Systems (OPODIS).* Studia Informatica, Paris, França, 143–162.

DOLEV, D., DWORK, C. E STOCKMEYER, L. 1987. Sobre a sincronia mínima necessária para o consenso distribuído. *J. ACM* 34,1 (janeiro), 77–97.

DOLEV, D., KRAMER, S., E MALKI, D. 1993. A entrega antecipada ordenou totalmente o multicast em ambientes assíncronos. Em *Anais do 23º Simpósio Internacional de Computação Tolerante a Falhas (FTCS-23)* (Toulouse, França). IEEE Computer Society Press. 544-553.

DOLEV, D. E MALKHI, D. 1994. O projeto do sistema Transis. Em *Theory and Practice in Dis tributed Systems,* K. Birman, F. Mattern e A. Schiper, Eds. Lecture Notes in Computer Science, vol. 938. Springer-Verlag, 83-98.

DOLEV, D. E MALKHI, D. 1996. A abordagem Transis para comunicação de cluster de alta disponibilidade. *com. ACM 39,* 4 (abril), 64–70.

DWORK, C., LYNCH, NA, E STOCKMEYER, L. 1988. Consenso na presença de sincronia parcial. *J. ACM 35,* 2 (abril), 288–323.

EKWALL, R., SCHIPER, A., E , P. 2004. Token transmissão atômica baseada em URBAN usando detectores de falha não confiáveis. In *Proceedings of 23nd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)* (Florianópolis, Brasil). IEEE Computer Society Press. 52–65.

EZHILCHELVAN, PD, MACEDO, RA E SHRIVASTAVA, SK 1995. Newtop: Um protocolo de comunicação de grupo tolerante a falhas. In *Proceedings of 15th IEEE International Conference on Distributed Computing Systems (ICDCS-15)* (Vancouver, Canadá). IEEE Computer Society Press. 296-306.

FEKETE, A. 1993. Modelos formais de serviços de comunicação: Um estudo de caso. *Computação IEEE. 26,* 8 (agosto), 37–47.

BLACK, A., LYNCH, N., E SHVARTSMAN, A. 2001. Especificando e usando um serviço de comunicação de grupo particionável. *ACM Trans. Comput. Sist. 19, 2* (maio), 171–216.

FELBER, P. E PEDONE, F. 2002. Transmissão atômica probabilística. In *Proceedings of 21st IEEE International Symposium on Reliable Dis tributed Systems (SRDS'02)* (Osaka, Japão). IEEE Computer Society Press. 170–179.

FELBER, P. E SCHIPER, A. 2001. Replicação ativa otimista. In *Proceedings of 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21)* (Phoenix, AZ). IEEE Computer Society Press. 333–341.

FETZER, C. 2003. Detecção de falha perfeita em sistemas assíncronos temporizados. *IEEE Trans. Com colocar.* 52,2 (fevereiro), 99–112.

FISCHER, MJ, LYNCH, NA, E PATERSON, MS 1985. Impossibilidade de consenso distribuído com um processo defeituoso. *J. ACM 32,* 2 (abril), 374–382.

FRIEDMAN, R. E VAN RENESSE, R. 1997. Mensagens de empacotamento como uma ferramenta para aumentar o desempenho de protocolos de ordenação total. In *Proceedings of 6th IEEE Symposium on High Performance Distributed Computing* (Portland, OR). IEEE Computer Society Press. 233–242.

FRITZKE , U. , INGELS , P. , MOSTEFAOUI , A. , E RAYNAL , M. 2001. Multicast tolerante a falhas baseado em consenso para tal ordem. *IEEE Trans. Paralelo. Distribuição. Sist.* 12,2 (fevereiro), 147–156.

GARCIA-MOLINA, H. E SPAUSTER, A. 1989. Ordenação de mensagens em ambiente multicast. In *Proceedings of 9th IEEE International Conference on Distributed Computing Systems (ICDCS-9)* (Newport Beach, CA). IEEE Computer Society Press. 354–361.

GARCIA-MOLINA, H. E SPAUSTER, A. 1991. Comunicação multicast ordenada e confiável. *ACM Trans. Comput. Sist.* 9, 3 (agosto), 242–271.

GOPAL, A. E TOUEG, S. 1989. Transmissão confiável em ambiente síncrono e assíncrono. In *Proceedings of 3rd International Workshop on Distributed Algorithms (WDAG'89)* (Nice, França). Notas de aula em Ciência da Computação, vol. 392. Springer-Verlag. 111-123.

GOPAL, A. E TOUEG, S. 1991. Inconsistência e contaminação. In *Proceedings of 10th ACM Symposium on Principles of Distributed Com puting (PODC-10).* Imprensa ACM. 257–272.

GUERRAOUI, R. 1995. Revisitando a relação entre compromisso atômico não bloqueador e consenso. In *Proceedings of 9th International Workshop on Distributed Algorithms (WDAG'95)* (Le Mont-St-Michel, França). Notas de aula em Ciência da Computação, vol. 972. Springer-Verlag. 87–100.

GUERRAOUI, R. E SCHIPER, A. 1997. Genuine atomic multicast. In *Proceedings of 11th International Workshop on Distributed Algorithms (WDAG'97)* (Saarbrucken, Alemanha). Notas de aula em Ciência da Computação, vol. 1320. Springer Verlag. 141–154.

GUERRAOUI, R. E SCHIPER, A. 2001. Multicast atômico genuíno em sistemas distribuídos assíncronos. *Teor. Comput. ciência 254,* 297-316.

GUY, RG, POPEK, GJ, E PAGE JR., TW 1993. Algoritmos de consistência para replicação otimista. In *Proceedings of 1st IEEE International Conference on Network Protocols (ICNP'93)* (São Francisco, Califórnia). IEEE Computer Society Press.

HADZILACOS, V. E TOUEG, S. 1994. Uma abordagem modular para transmissões tolerantes a falhas e problemas relacionados. TR 94-1425, Dept. of Computer Science, Cornell University, Ithaca, NY (maio.)

HILTUNEN, MA, SCHLICHTING, RD, HAN, X., CARDOZO, MM, E DAS, R. 1999. Canais confiáveis em tempo real: Personalizando atributos de QoS para sistemas distribuídos. *IEEE Trans. Paralelo. Dis trib. Sist.* 10,6 (junho), 600–612.

JALOTE, P. 1998. Transmissão ordenada eficiente em redes CSMA/CD confiáveis. In *Proceedings of 18th IEEE International Conference on Distributed Computing Systems (ICDCS-18)* (Amsterdã, Holanda). IEEE Computer Society Press. 112–119.

JIA, X. 1995. Um protocolo multicast de ordenação total usando árvores de propagação. *IEEE Trans. Paralelo. Dist. Sist.* 6,6 (junho), 617–627.

KAASHOEK, MF E TANENBAUM, AS 1996. Uma avaliação do sistema de comunicação do grupo Amoeba. In *Proceedings of 16th IEEE International Conference on Distributed Computing Systems (ICDCS-16)* (Hong Kong). IEEE Computer Society Press. 436–447.

KEIDAR, I. E DOLEV, D. 2000. Transmissão totalmente ordenada em face de partições de rede. Em *Computação de Rede Confiável,* DR Avresky, Ed. Kluwer Academic Pub., Capítulo 3, 51–75.

KEMME, B. E ALONSO, G. 2000. Não seja preguiçoso, seja consistente: Postgres-r, uma nova maneira de implementar replicação de banco de dados. In *Proceedings of 26th International Conference on Very Large Data Bases (VLDB 2000)* (Cairo, Egito). Morgan Kaufmann. 134–143.

KEMME, B., PEDONE, F., ALONSO, G. E SCHIPER, A. 1999. Processamento de transações em protocolos de transmissão atômica otimistas. In *Proceedings of 19th IEEE International Conference on Distributed Computing Systems (ICDCS-19)* (Austin, Texas). IEEE Computer Society Press. 424–431.

KEMME, B., PEDONE, F., ALONSO, G., SCHIPER, A. E WIESMANN, M. 2003. Usando transmissão atômica otimista em sistemas de processamento de transações. *IEEE Trans. Saber. Eng. de dados 15,* 4 (julho), 1018–1032.

KIM, J. E KIM, C. 1997. Um protocolo de ordenação total usando um esquema dinâmico de passagem de token. *Dist. Sist. Eng. J.* 4,2 (junho), 87–95.

KOPETZ, H., GRUNSTEIDL¨ , G., E REISINGER, J. 1991. Serviço de associação tolerante a falhas em um sistema de tempo real distribuído síncrono. In *Pro ceedings of 2nd IFIP International Working Conf. em Computação Confiável para Aplicações Críticas (DCCA-1)* (Tucson, AZ). A. Avizienis ÿ e J.-C. Laprie, Eds. Springer-Verlag. 411-429.

LAMPORT, L. 1978a. A implementação de sistemas multiprocessos distribuídos confiáveis. *Comput. Netw.* 2, 95-114.

LAMPORT, L. 1978b. Tempo, relógios e a ordem dos eventos em um sistema distribuído. *com. ACM* 21,7 (julho), 558–565.

LAMPORT, L. 1984. Usando tempo em vez de time-outs em sistemas tolerantes a falhas. *ACM Trans. Programa. Lang. Sist.* 6, 2, 256-280.

LAMPORT, L. 1986a. O problema da exclusão mútua: Parte I — uma teoria da comunicação entre processos. *J. ACM 33,* 2 (abril), 313–326.

LAMPORT, L. 1986b. Sobre a comunicação entre processos. parte I: formalismo básico. *Dist. Com colocar.* 1, 2, 77-85.

LAMPORT, L., SHOSTAK, R., E PEASE, M. 1982. O problema dos generais bizantinos. *ACM Trans. Programa. Lang. Sist.* 4, 3, 382–401.

LE LANN, G. E BRES, G. 1991. Transmissão atômica confiável em sistemas distribuídos com falhas de omissão. *Operador ACM Sist. Rev. SIGOPS 25,* 2 (abril), 80–86.

LIU, X., VAN RENESSE, R., BICKFORD, M., KREITZ, C., E CONSTABLE, R. 2001. Comutação de protocolo: Explorando meta-propriedades. Nos *Anais da 21ª Conferência Internacional IEEE sobre Oficinas de Sistemas de Computação Distribuída (ICDCSW'01), Intl. Workshop on Applied Reliable Group Com munication (WARGC 2001)* (Mesa, AZ). IEEE Computer Society Press. 37–42.

LUAN, S.-W. E GLIGOR, VD 1990. Um protocolo tolerante a falhas para transmissão atômica. *IEEE Trans. Paralelo. Dist. Sist.* 1,3 (julho), 271–285.

LYNCH, 1996. *Algoritmos Distribuídos.* Sr. Kaufmann, San Francisco, CA.

LYNCH, NA AND TUTTLE, MR 1989. Uma introdução aos autômatos de entrada/saída. *CWI Quarterly* 2, 3 (setembro), 219–246.

MALHIS, LM, SANDERS, WH E SCHLICHTING, RD 1996. Avaliação de desempenho numérico de um protocolo multicast de grupo. *Dist. Sist. Eng. J.* 3, 1 (março), 39–52.

MALLOTH, CP 1996. Concepção e implementação de um kit de ferramentas para a construção de aplicações distribuídas tolerantes a falhas em redes de grande escala. Tese de doutorado, Ecole Polytechnique F ed´erale de Lausanne, Suíça. Número 1557.

MALLOTH, CP, FELBER, P., SCHIPER, A. E WILHELM, U. 1995. Phoenix: Um kit de ferramentas para construir aplicativos distribuídos tolerantes a falhas em grande escala. No *Workshop de Plataformas Paralelas e Distribuídas em Produtos Industriais; 7º Simpósio IEEE sobre Processamento Paralelo e Distribuído.* San Antonio, Texas.

MAYER, E. 1992. Uma estrutura de avaliação para protocolos de ordenação multicast. In *Proceedings of ACM International Conference on Applications, Technologies, Architecture, and Protocols (SIGCOMM)* (Baltimore, Maryland). Imprensa ACM. 177–187.

MINET, P. E ANCEAUME, E. 1991a. ABP: Um protocolo de transmissão atômica. TR 1473, INRIA (junho). Roc Quencourt, França.

MINET, P. E ANCEAUME, E. 1991b. Ampla transmissão atômica em uma fase. *Operador ACM Sist. Rev. 25,* 2 (abril), 87–90.

MISHRA, S., PETERSON, LL, E SCHLICHTING, RD 1993. Consul: um substrato de comunicação para programas distribuídos tolerantes a falhas. *Dist. Sist. Eng. J.* 1, 1, 87–103.

MOSER, LE E MELLIAR-SMITH, PM 1999. Algoritmos de ordenação total resistentes a bizantinos. *Inf. Comput. 150,* 1 (abril), 75–111.

MOSER, LE, MELLIAR-SMITH, PM, AGRAWAL, DA, BUDHIA, RK, E LINGLEY-PAPADOPOULOS, CA 1996. Totem: Um sistema de comunicação em grupo multicast tolerante a falhas. *com. ACM 39,* 4 (abril), 54–63.

MOSER, LE; MELLIAR-SMITH, PM; AGRAWALA , V. 1993. Algoritmos de ordenação totais tolerantes a falhas assíncronos. *SIAM J. Comput. 22,* 4 (agosto), 727–750.

MOSTEFAOUI, A. E RAYNAL, M. 2000. Transmissão atômica baseada em consenso de baixo custo. In *Proceedings of 7th IEEE Pacific Rim Symposium on De pendable Computing (PRDC'00)* (Los Angeles, CA). IEEE Computer Society Press. 45–52.

NAVARATNAM, S., CHANSON, ST, E NEUFELD, GW 1988. Comunicação de grupo confiável em sistemas distribuídos. In *Proceedings of 8th IEEE International Conference on Distributed Computing Systems (ICDCS-8)* (San Jose, CA). IEEE Computer Society Press. 439–446.

NEIGER, G. E TOUEG, S. 1990. Aumentando automaticamente a tolerância a falhas de algoritmos distribuídos. *J. Algorithms 11,* 3, 374–419.

NESTMANN, U., FUZZATI, R., E MERRO, M. 2003. Modelagem de consenso em um cálculo de processo. In *(CONCUR 2003) Teoria da Concorrência, 14ª Conferência Internacional* (Marselha, França). RM Amadio e D. Lugiez, Eds. Notas de aula em Ciência da Computação, vol. 2761. Springer-Verlag. 393–407.

NG, TP 1991. Transmissões encomendadas para grandes aplicações. In *Proceedings of 10th IEEE International Symposium on Reliable Distributed Systems (SRDS'91)* (Pisa, Itália). IEEE Computer Society Press. 188–197.

PEDONE, F. 2001. Aumentando o desempenho do sistema com protocolos distribuídos otimistas. *Computação IEEE. 34,* 12 (dezembro), 80–86.

PEDONE, F., GUERRAOUI, R., E SCHIPER, A. 1998. Explorando transmissão atômica em bancos de dados replicados. In *Proceedings of EuroPar (Eu roPar'98)* (Southampton, Reino Unido). Notas de aula em Ciência da Computação, vol. 1470. Springer-Verlag. 513–520.

PEDONE, F. E SCHIPER, A. 1998. Transmissão atômica otimista. In *Proceedings of 12th International Symposium on Distributed Computing (DISC'98)* (Andros, Grécia). Notas de aula em Ciência da Computação, vol. 1499. Springer-Verlag. 318–332.

PEDONE, F. E SCHIPER, A. 1999. Elenco amplo genérico. In *Proceedings of 13th International Symposium on Distributed Computing (DISC'99)* (Bratislava, República Eslovaca). Notas de aula em Ciência da Computação, vol. 1693. Springer-Verlag. 94–108.

PEDONE, F. E SCHIPER, A. 2002. Lidando com semântica de mensagens com protocolos genéricos de broadcast. *Dist. Comput. 15,* 2, 97–107.

PEDONE, F. E SCHIPER, A. 2003. Transmissão atômica otimista: um ponto de vista pragmático. *Teor. Com colocar. ciência 291,* 79–101.

PEDONE, F., SCHIPER, A., URBAN´ , P., E CAVIN, D. 2002. Resolvendo problemas de acordo com oráculos de ordenação fracos. Nos *Anais da 4ª Conferência Europeia de Computação Confiável (EDCC-4)* (Toulouse, França). Notas de Palestra em Ciência da Computação, vol. 2485. Springer-Verlag. 44-61.

PETERSON, LL, BUCHHOLZ, NC, E SCHLICHTING, RD 1989. Preservando e usando o contexto na formação na comunicação entre processos. *ACM Trans. Comput. Sist.* 7, 3, 217–246.

POLEDNA, S. 1994. Determinismo de réplica em sistemas de tempo real distribuídos: Uma breve pesquisa. *Sist. Tempo Real.* 6, 3 (maio), 289–316.

RABIN, M. 1983. Generais bizantinos randomizados. In *Proceedings of 24th Annual ACM Symposium on Foundations of Computer Science (FOCS)* (Tucson, AZ). Imprensa ACM. 403–409.

RAJAGOPALAN, B. E MCKINLEY, P. 1989. Um protocolo baseado em token para comunicação multicast ordenada e confiável. In *Proceedings of 8th IEEE International Symposium on Reliable Distributed Systems (SRDS'89)* (Seattle, WA). IEEE Computer Society Press. 84-93.

REITER, MK 1994. Protocolos de acordo seguro: Multicast de grupo atômico e confiável em Rampart. Nos *Anais da 2ª ACM Conf. em Segurança de Computadores e Comunicações (CCS-2)* (Fairfax, VA). Imprensa ACM. 68–80.

REITER, MK 1996. Distribuindo confiança com o kit de ferramentas Rampart. *com. ACM 39,* 4 (abril), 71–74.

RODRIGUES, L., FONSECA, H., E VER´ÍSSIMO, P. 1996. Multicast totalmente ordenado em sistemas de grande escala. In *Proceedings of 16th IEEE International Conference on Distributed Computing Systems (ICDCS-16)* (Hong Kong). IEEE Computer Society Press. 503–510.

RODRIGUES, L., GUERRAOUI, R., E SCHIPER, A. 1998. Multicast atômico escalável. Nos *Anais da 7ª Conferência Internacional IEEE sobre Comunicações e Redes de Computadores* (Lafayette, LA). IEEE Computer Society Press. 840-847.

RODRIGUES, LT E RAYNAL, M. 2000. Transmissão atômica em sistemas distribuídos assíncronos de recuperação de falhas. In *Proceedings of 20th IEEE International Conference on Distributed Computing Systems (ICDCS-20)* (Taipei, Taiwan). IEEE Computer Society Press. 288-295.

RODRIGUES, LT E VER´ÿSSIMO, P. 1992. xAMP: um serviço de comunicação em grupo multi-primitivo. Nos *Anais do 11º Simpósio Internacional IEEE em Sistemas Distribuídos Confiáveis (SRDS'92)* (Houston, TX). IEEE Computer Society Press. 112–121.

RODRIGUES, LT, VER´ÍSSIMO, P., E CASIMIRO, A. 1993. Usando transmissão atômica para implementar um acordo a posteriori para sincronização de relógio. Nos *Anais do 12º IEEE Internacional Simpósio sobre Sistemas Distribuídos Confiáveis (SRDS'93)* (Princeton, NJ). IEEE Computer Society Press. 115–124.

SCHIPER, A. E RAYNAL, M. 1996. Da comunicação em grupo às transações em sistemas distribuídos. *com. ACM 39,* 4 (abril), 84–87.

SCHNEIDER, FB 1990. Implementando serviços tolerantes a falhas usando a abordagem de máquina de estado: um tutorial. *Computação ACM. Sobreviver 22,* 4 (dezembro), 299–319.

SHIEH, S.-P. E HO, F.-S. 1997. Um comentário sobre "um protocolo multicast de ordenação total usando árvores de propagação". *IEEE Trans. Paralelo. Dist. Sist.* 8, 10 (out.), 1084.

SHRIVASTAVA, SK 1994. Para CATOCS ou não para CATOCS, isso é... *ACM Operat. Sist. Rev. 28,* 4 (outubro), 11–14.

SOUSA, A., PEREIRA, J., MOURA, F., AND OLIVEIRA, R. 2002. Ordem total otimista em obras de rede de área ampla. In *Proceedings of 21st IEEE International Symposium on Reliable Distributed Systems (SRDS'02)* (Osaka, Japão). IEEE Computer Society Press. 190–199.

TOINARD, C., FLORIN, G., E CARREZ, C. 1999. Um método formal para provar propriedades de ordenação de algoritmos multicast. *Operador ACM Sist. Rev. 33,* 4 (outubro), 75–89.

URBAN´ , P., DEFAGO , X., E SCHIPER, A. 2000. Métricas com reconhecimento de contenção para algoritmos distribuídos: Comparação de algoritmos de transmissão atômica. Nos *Anais da 9ª Conferência Internacional IEEE sobre Comunicações e Redes de Computadores (IC3N'00)* (Las Vegas, NV). IEEE Computer Society Press. 582-589.

URBAN´ , P., SHNAYDERMAN, I., E SCHIPER, A. 2003. Comparação de detectores de falha e associação de grupo: Estudo de desempenho de dois algoritmos atômicos de transmissão ampla. In *Proceedings of IEEE International Conference on Dependable Systems and Networks (DSN'03)* (San Francisco, CA). IEEE Computer Society Press. 645–654.

VER´ÿSSIMO, P., RODRIGUES, L., AND BAPTISTA, M. 1989. AMp: Um protocolo multicast atômico altamente paralelo. *Comput. com. Rev. 19,* 4 (setembro), 83–93.

VICENTE, P. E RODRIGUES, L. 2002. Um algoritmo de pedido total uniforme indulgente com entrega otimista. In *Proceedings of 21st IEEE International Symposium on Reliable Distributed Systems (SRDS'02)* (Osaka, Japão). IEEE Computer Society Press. 92–101.

WHETTEN, B., MONTGOMERY, T., E KAPLAN, S. 1994. Um protocolo multicast totalmente ordenado de alto desempenho. Em *Teoria e Prática em Sistemas Distribuídos* (Castelo Dagstuhl, Alemanha). KP Birman, F. Mattern e A. Schiper, Eds. Lec ture Notes in Computer Science, vol. 938. Springer-Verlag. 33-57.

WIESMANN, M., PEDONE, F., SCHIPER, A., KEMME, B., E ALONSO, G. 2000. Entendendo a replicação em bancos de dados e sistemas distribuídos.

In *Proceedings of 20th IEEE International Conference on Distributed Computing Systems (ICDCS-20)* (Taipei, Taiwan). IEEE Computer Society Press. 264–274.

WILHELM, U. E SCHIPER, A. 1995. Uma hierarquia de multicasts totalmente ordenados. In *Proceedings of 14th IEEE International Symposium on Reliable Sistemas Distribuídos (SRDS'95)* (Bad Neuenahr, Alemanha). IEEE Computer Society Press. 106-115.

ZHOU, P. E HOOMAN, J. 1995. Especificação formal e verificação composicional de um protocolo de transmissão atômica. *Sistema em tempo real.* 9, 2 (setembro), 119–145.

Recebido em setembro de 2000; revisado em agosto de 2003; aceito em julho de 2004